AD-A098 016 GENERAL ELECTRIC CO SYRACUSE N Y F/G 9/2
SELF-DIAGNOSING DESIGN TECHNIQUES.(U)
JUN 78 R W HECKELMAN, W W KNIGHT, W W STRAUB F33615-77-C-1106
UNCLASSIFIED AFAL-TR-78-183 NL

1 OF 2
AD A
098 016

AFAL-TR- -183

# SELF-DIAGNOSING DESIGN TECHNIQUES

General Electric Company
P.O. Box 4840
Syracuse, N. Y. 13221

November 1978

TECHNICAL REPORT AFAL-TR-78-183

Final Report for 15 June 1977 - 30 June 1978

AIR FORCE AVIONICS LABORATORY
AIR FORCE WRIGHT AERONAUTICAL LABORATORIES
Air Force Systems Command
Wright-Patterson Air Force Base, Ohio 45433

[illegible]
J.B. RAWLINGS, II, Captain, USAF
Project Engineer

[illegible]
MELVIN ST. JOHN, Chief
Design & Packaging Group
Microelectronics Branch

*FOR THE COMMANDER:*

[illegible]
STANLEY E. WAGNER, Chief
Microelectronics Branch
Electronic Technology Division

SECURITY CLASSIFICATION OF THIS PAGE (When Data Entered)

| REPORT DOCUMENTATION PAGE | | READ INSTRUCTIONS BEFORE COMPLETING FORM |
|---|---|---|
| 1. REPORT NUMBER<br>AFAL-TR-78-183 | 2. GOVT ACCESSION NO.<br>D-A098 016 | 3. RECIPIENT'S CATALOG NUMBER |
| 4. TITLE (and Subtitle)<br>SELF-DIAGNOSING DESIGN TECHNIQUES. | | 5. TYPE OF REPORT & PERIOD COVERED<br>Final Report. |
| | | 6. PERFORMING ORG. REPORT NUMBER |
| 7. AUTHOR(s)<br>R.W. Heckelman, W.W. Knight, W.W. Straub | | 8. CONTRACT OR GRANT NUMBER(s)<br>F33615-77-C-1106 |
| 9. PERFORMING ORGANIZATION NAME AND ADDRESS<br>General Electric Co.<br>P.O. Box 4840<br>Syracuse, N.Y. 13221 | | 10. PROGRAM ELEMENT, PROJECT, TASK AREA & WORK UNIT NUMBERS<br>FY1175-77-21337 |
| 11. CONTROLLING OFFICE NAME AND ADDRESS<br>United States Air Force (AFWAL/AADE)<br>AFSC Aeronautical Systems Division<br>Wright-Patterson AFB, Ohio 45433 | | 12. REPORT DATE<br>JUNE 30, 1978 |
| | | 13. NUMBER OF PAGES<br>161 |
| 14. MONITORING AGENCY NAME & ADDRESS(if different from Controlling Office) | | 15. SECURITY CLASS. (of this report)<br>UNCLASSIFIED |
| | | 15a. DECLASSIFICATION/DOWNGRADING SCHEDULE |

16. DISTRIBUTION STATEMENT (of this Report)

"Approved for public release; distribution unlimited".

17. DISTRIBUTION STATEMENT (of the abstract entered in Block 20, if different from Report)

18. SUPPLEMENTARY NOTES

19. KEY WORDS (Continue on reverse side if necessary and identify by block number)

Self-Diagnosing, Fault Tolerant, Fly-By-Wire Flight Control, Self-Checking, Redundancy, Coding, Replication Error Management, LSI Fault Patterns,

20. ABSTRACT (Continue on reverse side if necessary and identify by block number)

A study and analysis of architectures, functional partitioning, and module and component features, required to achieve self-diagnosing microprogrammable capabilities in processors, is described. The results of the self-diagnosing techniques study, which apply to large-scale integrated (LSI) circuits, are summarized in a set of design guidelines. Application of these guidelines to a selected baseline, a digital fly-by-wire aircraft flight control system, has resulted in the design of a self-diagnosing, fault tolerant

DD FORM 1 JAN 73 1473 EDITION OF 1 NOV 65 IS OBSOLETE

SECURITY CLASSIFICATION OF THIS PAGE (When Data Entered) processor possessing "failed op$^{2}$" fault tolerance and a probability of failure $> 1 \times 10^{-9}$ for a two-hour mission.

[illegible]

Triplication is employed throughout the design because of the requirement to achieve correct operation after the detection of a second error. Functional, complementary partitioning of the requirements leads to an implementation that can be matched to highly integrated devices. A bit-slice processor implements the application requirements, while the error processing and processor redundancy management is handled by a computer-on-chip (COC) family of devices. Specially designed self-checking checkers and partition interconnections devices provide comprehensive and extended error protection and interlocked reconfiguration control with respect to both consistent and inconsistent errors.

Performance of the self-diagnosing, fault tolerant processor (SDFTP) is comparable to that of a simplex bit-slice processor. Throughput estimates indicate that instruction execution rates satisfy the baseline application requirements. Error recovery times depend on self-test diagnostic run times for some second error resolution and are currently projected to be less than 10 milliseconds.

Implementation of the SDFTP requires approximately 4.6 times as many devices as the simplex equivalent processor and nearly 40% added parts types, of which nearly all are LSI circuits.

A program plan to implement and demonstrate the self-diagnosing, fault tolerant processor is included.

## FOREWORD

The effort, which this following report documents, defines approaches for designing highly reliable airborne processors using, for the most part, present off-the-shelf hardware. Designs are intimately described for several special purpose chips to handle the voting and error handling tasks of the fault tolerant design. This program achieved the design of an airborne processor targeted for flight control applications with a probability of failure of less than 1 x $10^{-9}$ in a two hour mission. Throughput is in the vicinity of 300,000 to 400,000 operations per second. Error recovery would require a maximum of 10 milliseconds. CPU instruction set architecture is that of a presently available processor so that very little additional support software need be developed.

This report was submitted in October 1978.

Publication of this report does not constitute USAF approval of report findings or conclusions, and has been accomplished only for the exchange and stimulation of ideas.

TABLE OF CONTENTS

TABLE OF CONTENTS
(Continued)

## LIST OF ILLUSTRATIONS

LIST OF ILLUSTRATIONS
(Continued)

# I. INTRODUCTION

## A. GENERAL DESCRIPTION

The Self-Diagnosing fault tolerant processor (SDFTP) demonstrates that large scale integrated (LSI) circuit devices can be used to effectively implement current and future military avionic digital system requirements. In this way the predicted advantages of reduced size, weight and cost of LSI implementations are realized. The design also shows that commercially available LSI devices can be used to implement a large percentage of the processor. By supplementing these devices with a few special part types and using self diagnosing and fault-tolerant techniques, a processor has been designed with the required high fault tolerance and predicted reliability required for airborn fly-by-wire flight control processor application. Many of these special circuits should find use in other equipments having lower fault tolerance and reliability requirements, to the extent that their utilization could become widespread.

This high predicted reliability is achieved while taking into account the modifications to the fault models of current digital circuits which highly integrated devices require for high-confidence reliability predictions.

The design employs both dynamic and static redundancy in conjunction with self-diagnosing design techniques to produce the necessary reliability enhancement and fault-tolerance improvement of the simplex processor. Although redundancy is employed as the primary means of achieving fault tolerance, periodic self-testing is recommended for status assessment and initial flight check-out. Because of this approach maintenance should be processor directed using built in self-diagnoses.

Both bit-slice and monolithic microprocessors are used in a complementary fashion that matches the device capabilities to the functional division of the processor requirements. The division of labor between the processors is derived, primarily, from the performance and reliability requirements and the desire for a wide range of applicability. Assignment of the application functions to the bit-slice processor and the management of the fault tolerance requirements to the computer-on-the chip family of devices produces a design that is well matched to its intended use.

This report begins with a summary of the baseline application studies, which led to the selection of the fly-by-wire flight control application, and of the design guidelines developed for the design of self-diagnosing processors using LSI components. A summary of the self-diagnosing fault tolerant (SDFTP) design concludes this Section. Section II present conclusions and recommendations.

Section III, Processor Design is a detailed description of the design beginning with a general overview followed by a discussion of the bit-slice processor and the computer-on-chip implementation of the Reconfiguration and Recovery Unit. Section IV, Processor Self-Test, relates the self-test requirements to some preliminary simulation results in terms of test program size and coverage based on LSI device implementations. Reliability modeling and failure analysis of the SDFTP and the simplex processor are covered in Section V, Reliability Enhancement and Prediction.

SDFTP and the simplex processor are compared in Section VI, and Section VII discusses the plan for designing, building, and demonstrating a self-diagnosing, fault-tolerant processor. The two appendices detail the baseline requirements study and the design guidelines.

## B. APPLICATION SUMMARY

Two airborne applications were selected as potential sources of baseline requirements. The applications considered were the fly-by-wire flight control processor and the synthetic aperture ground map function of an airborne multimode radar signal processor. Both applications were examined to determine the requirements, beginning with mission identification and functional analysis. The work led to the development of algorithm flow followed by performance analysis of representative tasks and resource sizing, in terms of memory, processor speed and complexity as measured by the variety of operations and execution speed. These results are summarized in Appendix I,

### 1. Definition of Baseline Processing Requirements

The flight-control application is considered first in Section A of Appendix I since it represents a set of requirements that falls within the capabilities of systems that can be configured in a single programmable computer structure, and which can be implemented today using existing LSI devices such as microprocessors and memories. It is estimated that a high performance control configured, fly-by wire aircraft would require less than 16,000 words of 16-bit wide memory and could be controlled by a processor capable of executing instructions at a rate of 300 to 400 thousands of operations per second (KOPS). Because of the safety requirements of this application, quadruple redundancy coupled with software implemented redundancy management leads to a sophisticated input output system that connects the electronic flight control system to the aircraft control sensors and actuators. The reconfiguration approach selected is designed to achieve "failed op-squared" fault tolerance for the electronics.

The second application, Synthetic Aperture Ground Map Processing of a Multimode Radar (described in Appendix I), results in signal processor requirements that are beyond the capability of current and near future single conventional microprocessor designs. However, special programmable pipeline processors and netted sets of microprocessors are believed capable of achieving the performance required. As in many other radar signal processing applications, the core signal processing function has the ability to generate a doppler frequency analysis of the radar return. For this ground-mapping mode of the multimode radar, a processing rate in excess of $20 \times 10^6$ complex multiplies is required in addition to a signal processing operation rate in the 1-2 million instructions per second range. Compared to the flight-control application, the multimode memory requirements are significantly larger and are estimated to fall in the 3.5 million bit range.

This storage is normally distributed throughout the signal processor and must provide a high memory accessing rate capability, which is a function of the specific radar mode and signal processor architecture.

## C. DESIGN GUIDELINES

The effects of architecture, functional partitioning, and module and component features on microprogrammable self diagnosing capabilities of digital processors were investigated. These results were then used to create a set of design guidelines for designing self-diagnosing, fault-tolerant processors for both monolithic and bit-slice processors. Appendix II, Design Guidelines, covers the findings of the study and describes the techniques considered.

Application of the guidelines strongly influenced the design of the SDFTP. One of the major conclusions is that architectural considerations are of primary importance in the design of a self-diagnosing processor, particularly those making extensive use of large scale integrated (LSI) circuits. A major factor in this conclusion is that fault models of these self-diagnosing processors should include multiple errors. Applying this constraint in the evaluation of checking techniques leads, in general, to the selection of replication as the preferred coding approach and for processors in particular. Periodic testing is rejected as a primary approach because of its poor detection of inconsistent errors, incompatibility with real-time requirements, uncertain effectiveness of diagnostic routines in accounting for multiple errors, and difficulty in generating multiple error diagnostics.

Utilization of redundancy for self-diagnosis and fault tolerance leads to an increase in the probability of failure and to the desirability of enhancing the reliability of the self-diagnosing processor design. However, most of the redundancy techniques that are theoretically interesting apply only at the component level and, if applied as static redundancy, are effective for short times compared to their mean-time to failure. Consequently, bit-slice processors are preferred over monolithic microprocessors because of their lower circuit complexity and greater freedom in partitioning. Technological considerations dictate that redundancy be applied external to the device, at least for the immediate future.

Partitioning of the processor designs is based on hardware attributes, fault error models, and type of diagnosis. Hardware attributes include partition function, structure regularity, size, speed and communication requirements. Using these criteria, functional partitioning was determined to be the most effective type. For a bit-slice processor the partitions are: processor, control, memory, input output and buses. Placement of the boundaries of the partitions were strongly influenced by the breadth of communication required among the partitions since the size of the interface circuitry was a direct function of the number of interface signals.

The guideline summary for a fault tolerant, self-diagnosing bit-slice microprocessor is presented in Table I . The bus recommendation applies only to internal communication within the processor and memory and does not account for noise. Memories can also be an exception to the general recommendation of redundancy for self-diagnosing protection. For medium

TABLE I
GUIDELINE SUMMARY BY MICROPROCESSOR TYPE

BIT SLICE MICROPROCESSOR (16 BIT)

FAULT TOLERANT SYSTEM

| | REPLICATION LEVEL | | | | CONVENTIONAL CODE | | |
|---|---|---|---|---|---|---|---|
| PARTITION | TRIPLI-CATION & VOTE | DUPLI-CATION & COM-PARE | DUPLICATE/ STANDBY | SIMPLEX | ** SEC/DED | SELF-CHECKING | PARITY |
| PROCESSOR | X | X | | | | | |
| CONTROL | X | X | | | | | |
| MEMORY | SMALL MEMORY X | X | | | MED TO LG MEMORY X | MED TO LG MEMORY * X | |
| BUSES | | | | X | | | X |
| CLOCK | [illegible] | | X | | | X | |

* - PROTECTS MEMORY DECODERS AND ENCODERS

** SEC/DEC - SINGLE ERROR CORRECTIONS DOUBLE ERROR DETECTING CODE to large-sized memories conventional coding techniques are efficient and effective. Specifically, a single error correcting-double error detecting code with self-checking implementation of the encoders and decoders is recommended.

## D. SDFTP DESIGN

The SDFTP design is a synergistic combination of commercially available bit-slice microprocessors and computer-on-chip family of devices. It can be the archtype for a range of microprocessor-based systems, which meet high reliability military standards, using LSI devices. It employs self-diagnosing techniques, redundancy, and deferred maintenance to achieve a high level of fault tolerance — i.e., tolerate two faults with correct operation after the second fault. The design is developed according to a top-level functional partitioning of the requirements into two sets. They are the application set and the fault tolerant set. The application set is implemented by a replicated bit-slice microprocessor that handles the execution processes and communicates with the outside world.

The second set is concerned with the detection of errors and the management of the resources to achieve the required level of fault tolerance. It is implemented with a combination of special custom-designed devices and a computer-on-chip family of devices. This combination of devices performs the five functions of fault tolerance:

1) error detection
2) error location
3) failed function substitution
4) reconfiguration
5) recovery

This distribution of resources follows the Design Guidelines developed in the beginning of this program (See Appendix II Design Guidelines). It is a reflection of the observation that architecture is the most important factor in the design of a self-diagnosing system.

In highly reliable systems, such as this flight-control application, partitioning also ranks high because of the trade-off that must be achieved between access, partition failure rate, and communication path width. Functional partitioning of the bit slice microprocessor resulted in:

1) generality of application through microprogrammability

2) performance sufficient to satisfy the flight-control application

3) partition failure rates that were sufficiently low that replication could achieve the high reliability requirements of flight-control applications.

4) partition interfaces that had reasonable hardware coupling such that the necessary interface devices had relatively low failure rates in comparison with those of its partition

The special, custom devices were instrumental in achieving this last result. They are of a complexity, structure, and size that conventional tools can be effectively used to analyze their properties and have failure rates that are small compared to those of the partitions. They incorporate the ideas needed for effective testability through the incorporation of scanning registers and low level combinatorial networks between these registers.

Both the bit-slice microprocessor and the computer-on-chip (COC) are triplicated for five reasons:

1) Three copies of a function are needed, at a minimum, if "failed op$^2$" fault tolerance is needed.

2) It was found that triplication provided sufficient reliability enhancement for the short missions, so that a probability of failure of less than $1 \times 10^{-9}$ has been predicted.

3) Triplication is next to the cheapest of the single error codes that can be used to reliably detect the multiple bit errors exhibited by LSI devices.

4) Totally self-checking checkers (TSC's) can be designed and applied in such a way that any illegal input code (nonidentical triad of signals) as well as any checker failure can be detected. This extends the protection boundary of the bit-slice processor to the input buffers of the computer-on-chip.

5) By employing a triplicated cascade of these totally self-checking checkers in a tree structure, a malfunction transparent error collection network can be implemented. This network transforms the multiple self-checking checker outputs to a single signal that can interrupt the processing in the bit-slice processors and can alert the computer-on-chip that an error has been detected in the bit-slice processors.

A single string version of the SDFTP is shown in Figure 1. The bit-slice processor has been divided into three functional partitions resulting in three internal interfaces and one output (memory) interface. Each interface includes a pair of self-checking checkers (SCC) with triple-encoded outputs connected to the COC input buffers and the SCC tree. When an error occurs, the COC is interrupted by the output of the SCC tree and COC reads in the error information from the interfaces through the COC Input Buffer. Using this information the COC locates the error by interface and device and determines whether reconfiguration of the bit-slice processor is required. If it is, the COC sends a reconfiguration command to the appropriate bit-slice processor voter-switches located at each interface, by means of its Output Buffer. After reconfiguration has been completed, the COC initiates recovery on the bit-slice processor by interrupting it and supplying it with an interrupt vector, which, in most instances, is the address of the last roll-back point of the application process in execution at the time of the interrupt. If reconfiguration is not required after locating the error, the COC initiates the recovery process.

In both cases the COC saves the error information and updates its error history and SDFTP configuration status. This information can be made available during flight for status assessment and, when not in flight-control use, in computer-aided maintenance. This leads to improved repair and higher processor availability.

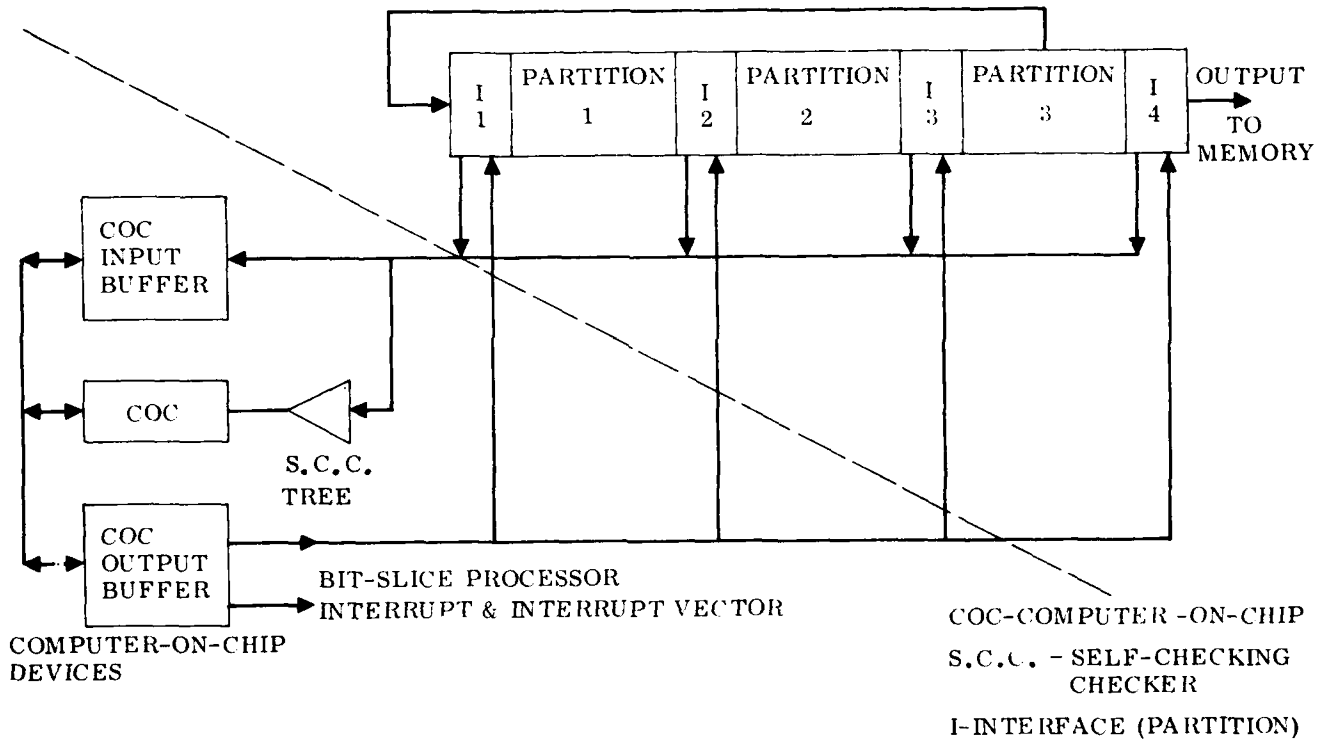


Figure 1. Bit-Slice Microprocessor

When the COC is not processing error reports, it performs self-test, reads out data to a status display, or exercises the SDFTP to determine if the error response system is operational.

Each interface between the bit-slice processor partitions consists of a fixed interconnection of devices that perform the following functions.

| | Function | Device |
|---|---|---|
| 1) | Check Partition Output | (Partition) Output SCC |
| 2) | Reconfigure Interface & Perform Failed Function Substitution | Voter-Switch |
| 3) | Check Voter-Switch Output | (Partition) Input SCC |

Interface interconnections are shown in Figure 2. Each interface has an output and input SCC function. The output SCC function is implemented by special custom devices, which perform the totally self-checking checker function by pairs of signals on the three partition outputs (COPY 1, COPY 2, COPY 3), and have three code outputs. The device also snapshots each triple input microcycle of the bit-slice processor. These inputs and the three TSC's can be read out to the COC on command. The three TSC outputs are also connected to the SCC trees shown in Figure 1.

The input SCC device is identical to the output SCC. It receives inputs from the Voter-Switch device. These Voter-Switch triple inputs are checked in pairs by connecting the three combinations of signals to be checked to the inputs of the three TSC's. The outputs of the TSC are also connected to both the COC input buffers and the SCC trees. Snapshots of the Voter-Switch outputs are also captured every microcycle and can be read out on command from the COC.

Reconfiguration and failed function substitution are the functions performed by the Voter-Switch. Each triple of input bits of the partition word can either be voted, or one of the three bits can be connected to the output by a three-way switch. The output can be controlled by the COC by sending a command that selects either the voter or the switch. If the switch connection is elected, the command selects one of the three positions of the switch so that the selected Voter-Switch input is connected to the output.

Two other special devices have been defined to make the fault tolerance more effective and efficient. They are the SCC With Mask and the Clock Controller. The SCC With Mask is used in the SCC tree. It has two features that distinguish it from the SCC Without Mask. They are a wider input capability (24 bits versus 16 bits) and an input masking capability. This mask capability permits an input to be blocked so that its signal does not contribute to the output of the device. It is used to prevent a faulted device from causing interrupts after it has failed.

The clock controller is used with the COC device to inhibit inadvertent reconfiguration of the bit-slice processor. This protection is implemented by interlocking the reconfiguration command clock using redundancy and key codes.

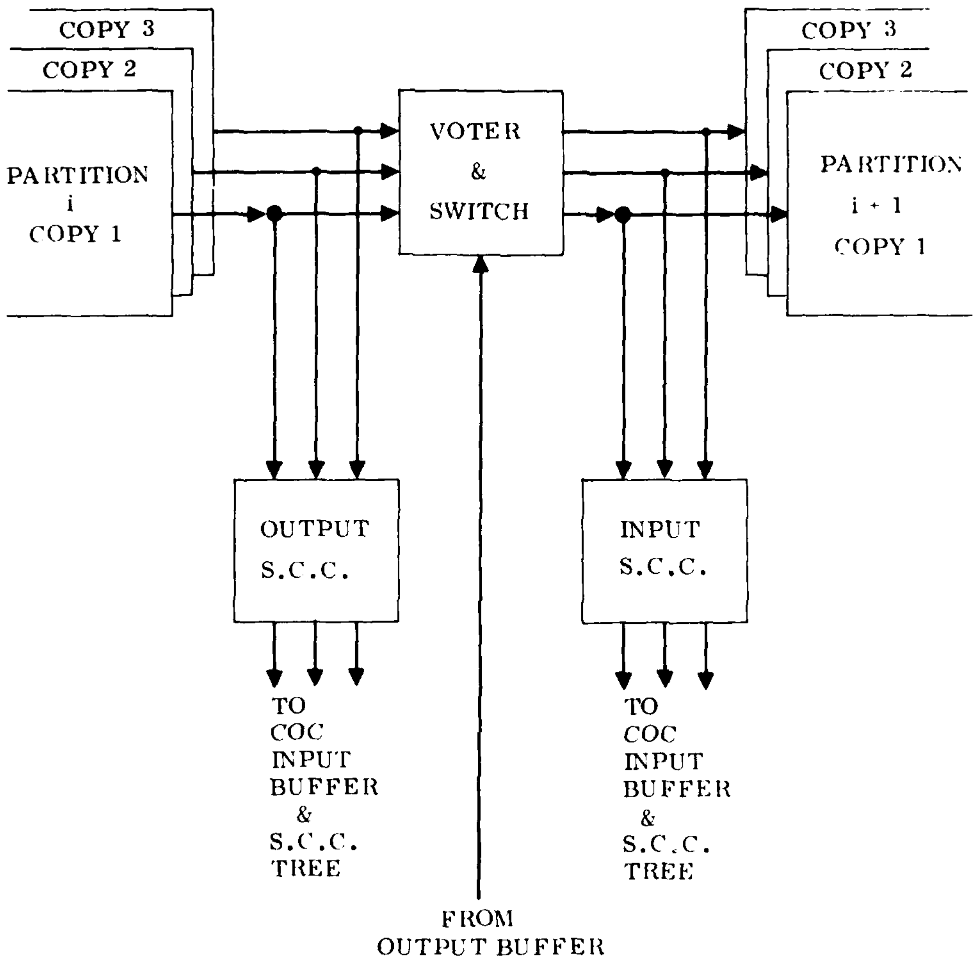


Figure 2. Interface Interconnections

The last special circuit is the microsequencer that replaces six devices in the microsequencer partition. Since the SDFTP design replicates this partition three times, the savings are tripled. In addition to these special considerations, the device has general appeal since it is believed to be a better match, to most of the applications that are likely to be encountered in dedicated military applications, than presently available microsequencer devices.

These devices and techniques result in a design of greatly improved reliability compared to the simplex processor. The SDFTP reliability is calculated to be four to five orders of magnitude better than the simplex processor for self-test coverage in the range of 0.9 to 1.0. These results are based on detailed models of the SDFTP, which have been conservatively established. The associated failure rates that were used in the model were computed using military handbook values or procedures. Where necessary, as is the case for custom devices and more complex commercially-available devices, projections of the failure rates were developed. These projections were based on an extrapolation of the complexity weights of the failure rate equations. For an assumed self-test coverage factor of .95, the probability of failure was determined to be less than $1 \times 10^{-9}$ for a three-hour mission.

Thus, compared to the simplex processor design, the SDFTP is significantly superior for high fault-tolerant and high-reliability applications with short mission time, such as the fly-by-wire electronic flight control processor. The SDFTP design can tolerate two faults and still provide undegraded operation, while the simplex processor has no tolerance at all. The testability improvement has not been demonstrated, but should also be much superior to the simplex design due to the partitioning and the insertion of the scanning registers Hence, the repair rate and availability should be much improved, since the design is self diagnosing and retains a run time history.

Another of the major improvements should be in the quality of the output because of the tolerance of the design to inconsistent errors. Transients should, in general, be masked until the occurrence of a second fault in a device. The value of this improvement is difficult to quantify but available data suggest that inconsistent errors can have frequencies of occurrence that are 10 times that of the solid errors.

Performance of this SDFTP in terms of execution rate and throughput closely parallels that of the simplex processor. The difference is believed to be sufficiently small so that it is negligible. Loss of throughput due to the processing of detected error is between 5 to 10 milliseconds per detected error with the average closer to five. This is due to the low probability of a second error in the same interface device.

The price for this improvement is, of course, an increase in physical resources. The SDFTP requires about 210 devices, which is about 4.6 times the number of devices needed to implement a simplex design. Most of the increase is due to the triplication requirement but the difference between the 4.6 factor of the SDFTP and triplication, 1.6, is a measure of the efficiency of the self-diagnosing self-checking checker and the dynamic redundancy control. The parts type comparison shows that eight additional devices are required, most of which are of the LSI variety. Four of these are special custom designs. At the cost of adding one additional device the total parts count can be reduced to about 195 devices by employing the microsequencer custom device.

For this commitment, the advantages of a very reliable, fault-tolerant, self-diagnosing LSI implemented design can be realized. Increasing the number of LSI devices and the degree of integration of the devices decreases the size, weight, and ultimately the cost. This approach enhances these advantages while accounting for the likelihood of LSI induced multiple-bit errors.

## II. CONCLUSIONS

A two fault tolerant, self-diagnosing, highly reliable microprocessor, capable of providing modest throughput,has been designed using design guidelines based on LSI devices. This microprocessor's execution speed, fault tolerance, and short mission reliability more than exceeds the selected fly-by-wire electronic flight control processor baseline requirements. The guidelines have yielded a design that tolerates the multiple bit error patterns that can accompany LSI device operation. Execution of the design has reinforced the design guideline conclusions: to wit, that architectural considerations are of primary importance in the design of a self-diagnosing microprocessor. Standardization of the "optimized" functional partition's interfaces has permitted the definition of just two custom LSI interface devices that implement the totally self-checking checkers and the Voter-Switch devices for all of the partition interfaces. This implementation has resulted in an interface implementation that has sufficiently high reliability that it does not appreciably degrade the partition reliability.

Top-level functional decomposition of the total requirements has permitted a bit-slice processor implementation of the application requirements using commercially available devices. An optional custom LSI circuit, which decreases the total parts count of the SDFTP by 15, has been defined. Execution of the other top-level functions by a computer-on-chip implementation results in error recovery processing times of the order of 5-10 milliseconds. These functions include the error vector pattern analysis for fault location and error management for reconfiguration and recovery. The average execution time should be in the 5-7 millisecond range, since the longer times correspond to the double error in the same interface design, which has a relatively low probability.

This SDFTP requires a commitment of about 4.6 times the simplex parts count. For some applications this may be too expensive,and other designs should be considered if the fault tolerance and/or reliability requirements are not as severe as in this application. Single fault tolerance appears to have appreciably lower parts count while retaining most of the testability features of the more tolerant design.

# III. SELF-DIAGNOSING FAULT TOLERANT PROCESSOR

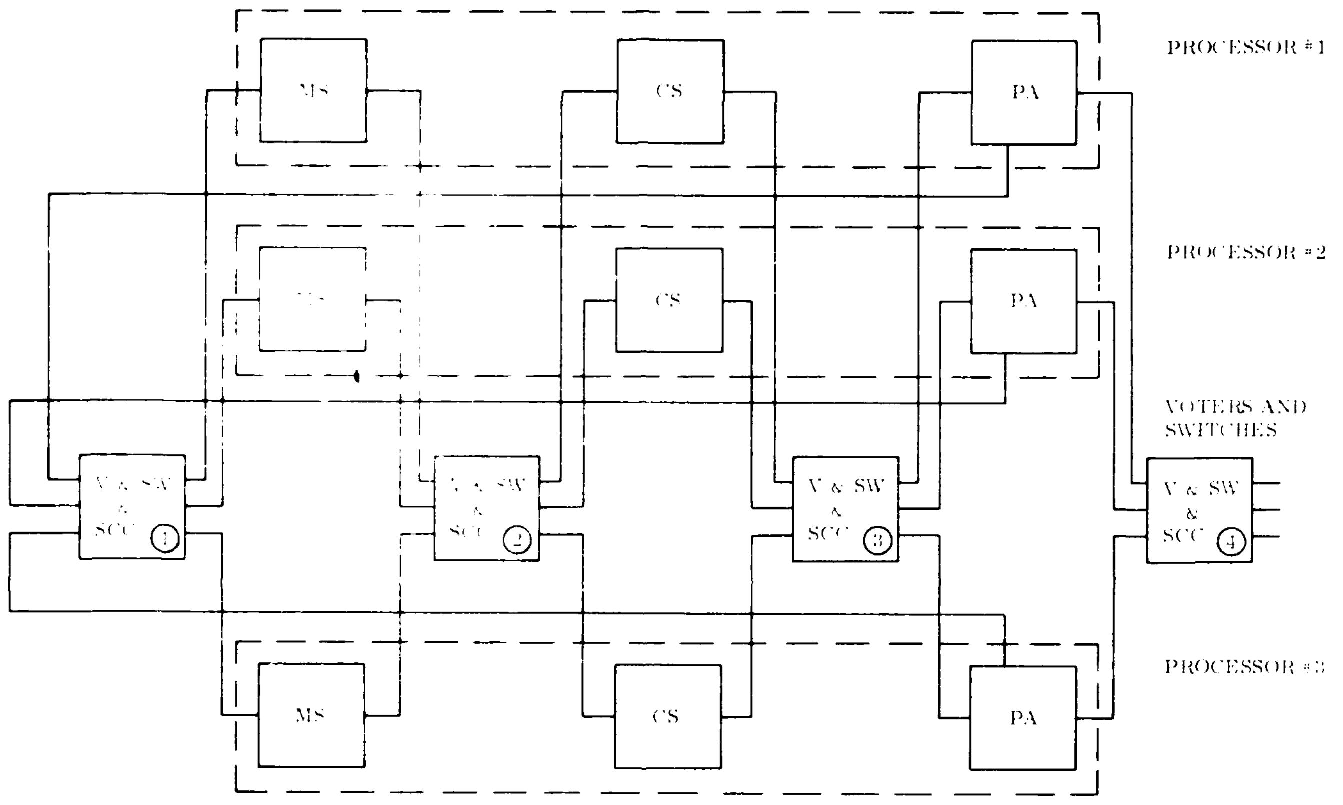
## A. GENERAL DESIGN CONSIDERATION AND OPERATION

The Self-Diagnosing Fault Tolerant Processor (SDFTP) design consists of triplicated structures. A triplicated bit-slice microprocessor, as shown in Figure 3, executes the flight control program while a triplicated monolithic microprocessor manages the bit-slice microprocessor's redundancy (see Figure 4). This Reconfiguration and Recovery Unit processes the error reports generated by the bit-slice processor checkers to determine how the bit-slice processor should be configured and the sequence of programs that the bit-slice processor should execute to return to normal processing.

Basically, a bit-slice processor was selected for the flight control processor because (1) it provided a better match to the throughput requirements of the flight control application, and (2) its lower level of integration permits smaller redundant structures to be implemented. The second attribute permits lower failure rate structures to be replicated, resulting in significantly more reliable designs for short missions. These smaller structures have been selected according to the guidelines presented in Appendix II. Accordingly, the bit-slice processor architecture was divided into three parts corresponding to the three functions of the processor hardware. See Figure 5. These partitions are the microsequencer, the control store-pipeline register, and the processor array, which includes the microprocessor bit slices. As shown in Figure 1, there are three internal and three external partition interfaces. All of the internal interfaces and the processor array-memory external interface include Voter-Switch devices for changing the interconnections between the pairs of partitions. Included in each of these interfaces is a pair of checkers for checking the output and input of the partitions involved in each partition as shown in Figure 6.

It is these checkers that detect any partition output errors and Voter-Switch errors at each interface. In addition, these checkers possess the very significant feature that they can detect their own errors. These self-checking checkers (S.C.C.) also extend the boundary of protection from the partitions and Voter-Switches to the periphery of the checkers. Each of the checkers receives an input from each of the copies of the partition or the Voter-Switch and develop an error signal corresponding to a check of each pair of signals. Since there are three possible combinations, each S.C.C. has three outputs corresponding to these three pairs of signals.

These S.C.C. error signals are used to alert the RRU that an error has been detected, and to locate the partition,Voter Switch or S.C.C. that has the error. Each of the Maskable S.C.C.'s shown in Figure 4 combines all of the error outputs from the bit-slice microprocessor S.C.C.'s and produces a dual-rail signal that interrupts its associated RRU computer. The interrupt causes each RRU computer to begin its diagnostic program to locate the error.



MS - MICROSEQUENCER
CS - CONTROL STORE
PA - PROCESSOR ARRAY
V &SW & SCC - VOTER & SWITCH & SELF-CHECKING CHECKER

Figure 3. Processor (Less RRU)

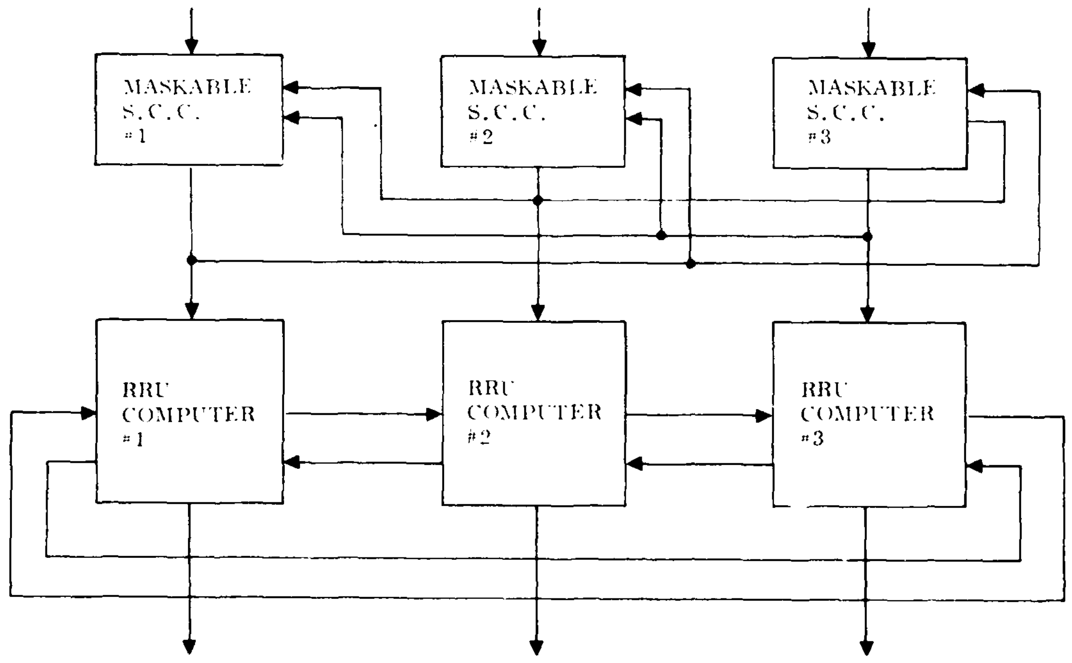


Figure 4. Triplex RRU

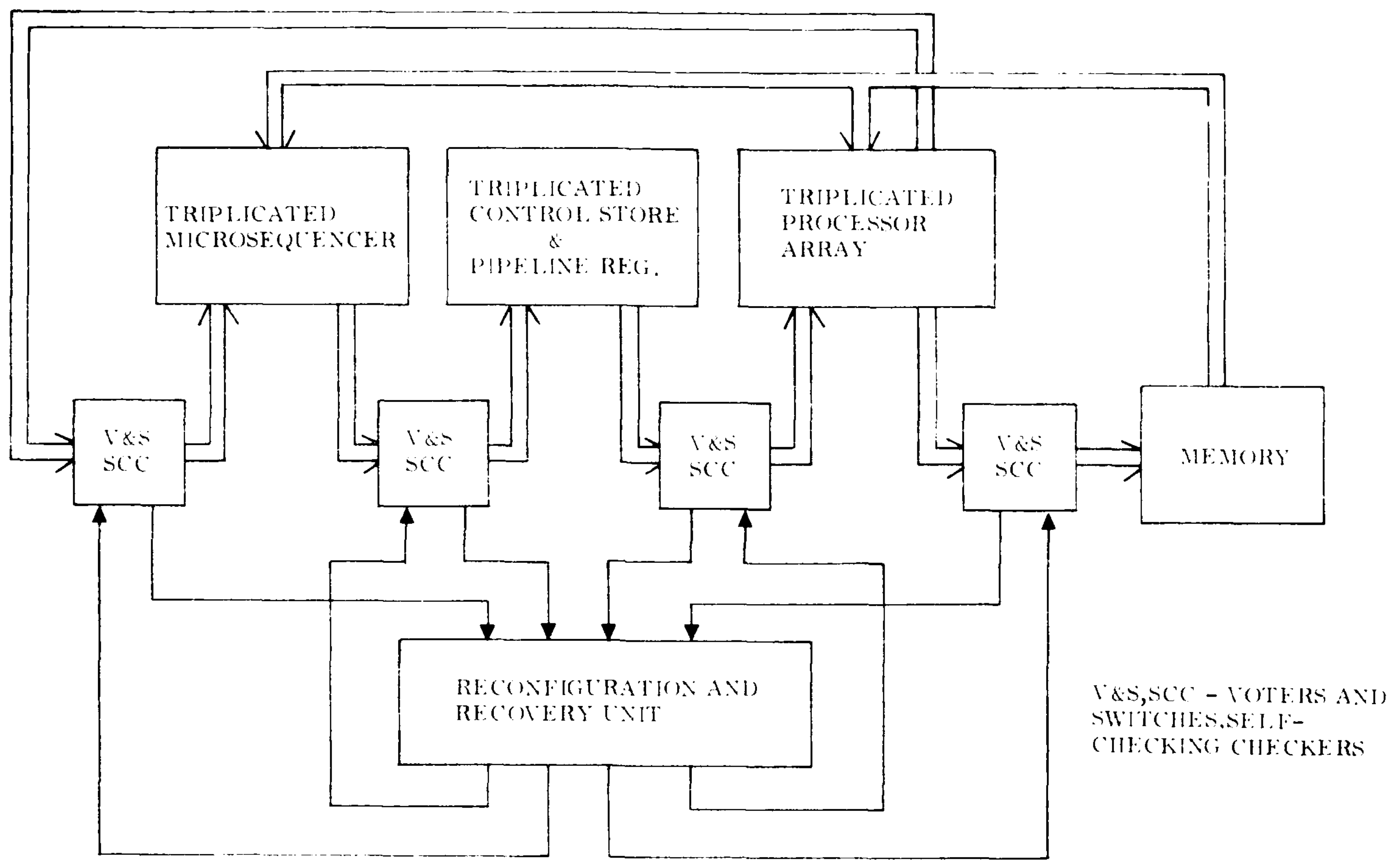
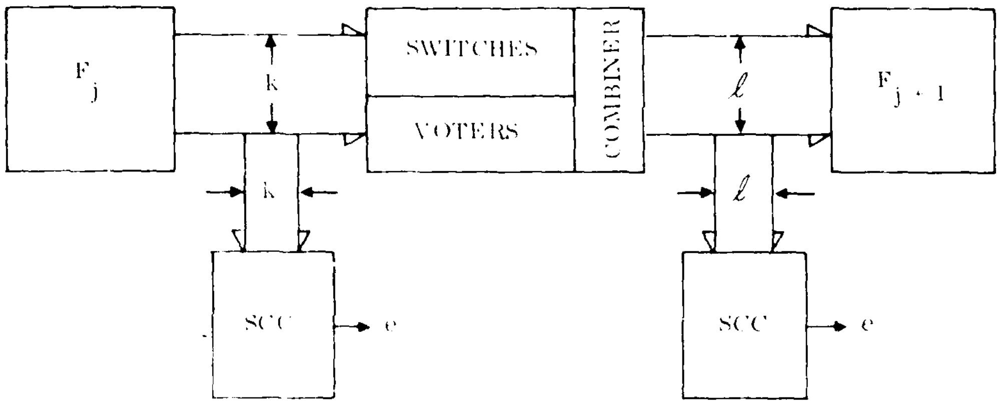

TRIPLICATED MICROSEQUENCER
TRIPLICATED CONTROL STORE & PIPELINE REG.
TRIPLICATED PROCESSOR ARRAY
V&S SCC
V&S SCC
V&S SCC
V&S SCC
MEMORY
RECONFIGURATION AND RECOVERY UNIT
V&S,SCC - VOTERS AND SWITCHES,SELF-CHECKING CHECKERS

Figure 5. Block Diagram SDFTP



VOTER & SWITCH

- ~ 350 GATES
- 64 PIN

Figure 6. Partition Interface Interconnection
(Voter and Switch and Self-Checking Checkers)

Simultaneously, the S.C.C. dual-rail signal interrupts the bit-slice processor clocks and prevents the error signal from propagating beyond the partition in which it originated. With the bit-slice processors stopped, the RRU's have time to examine all of the bit-slice processor's S.C.C. output signals, without concern that they might change. By reading the contents of one of the three snapshot registers in the S.C.C., each RRU computer can determine the source of the error and whether reconfiguration of the bit slice processors is required. Depending on the status of the SDFTP the RRU may read all or only a portion of the contents of the snapshot register. In some cases, the error vector associated with each S.C.C. may be sufficient. In other cases, the entire snapshot may be read out, including the current S.C.C. input vectors and the immediately preceding microcycle input vectors. By comparing these vectors, the failed partition or device can be located. In some second error situations, the comparison may require that a self-test program be run, in which case the comparison includes a predetermined value that is known to be correct. If the snapshot values differ from the precomputed value, their source is inferred to have failed.

Combining this information with the stored status of the SDFTP, the RRU computers determine the best configuration for the SDFTP and, if it is not the current one, determine the commands that must be issued to the Voter-Switches to produce this configuration. Each RRU controls one of three sets of voters and switches in each Voter-Switch device. By issuing the proper command, the RRU computer can select a configuration of one of three channels in each Voter-Switch. One of the commands selects a voter that produces the majority function of three inputs, corresponding to some bit of the interface data. The other three commands select one of three switches. Each switch connects one of its three inputs to an output for each bit of the interface signal.

The recommended mode of operation is to use the voter through the occurrence of the second error in the same partition or voter switch in order to retain the masking capability of the voter for as long as possible. Hence, the operating algorithm assumes that the first error is an inconsistent one, since the expected frequency is considerably higher than that for consistent (solid) faults; it also assumes that the error will not recur when the program is rolled back to the last roll-back point. Solid errors of course will be detected again at the same point in the program and the system will then be reconfigured using the switches to select an input signal. Each channel will use a different switch so that the outputs are independent insofar as possible. The failed channel will have to use one of the remaining two channels that are still good.

After an error has been confirmed as being consistent, the Maskable S.C.C.'s are modified so that the S.C.C. output producing the error is rendered ineffective in controlling the RRU computer and clock interrupt signals. This is achieved by allowing the RRU computers to generate a mask that blocks out only the signal inputs produced by the error to the Maskable S.C.C.'s.

Once reconfiguration is completed, recovery is initiated. For all of the first errors in a device or partition, recovery consists of turning the bit-slice processor clocks on and vectoring the bit-slice processors to the last rollback point in the executing task. The same procedure is followed for all second faults that do not occur in the same device or partition. However, when the error occurs in the same partition or the same part of the Voter-Switch, the recovery process requires the execution of a self-test program to identify which of two partitions is faulted. If the self-test program does not result in the detection of an error, the bit-slice processor automatically falls through the last roll-back point. If another error located in the same place is detected, the SDFTP is reconfigured and, then, the bit-slice processors are vectored back to the last roll-back point by the RRU computers.

When the RRU computers are not processing bit-slice processor S.C.C. error reports, they are executing self-test and cross-checking each other. If the RRU Maskable S.C.C. fails, the associated RRU computer switches to one of the other two Maskable S.C.C. inputs. In some instances, this will be ineffective and the RRU must reconfigure itself just as it must when an RRU computer fails. To prevent inadvertent changing of the Voter-Switch commands, the shift clocks associated with these commands are interlocked by a Clock Controller device that requires a key (code word) and/or the majority of the RRU computers to command a clock signal.

## B. PROCESSOR DESIGN VERIFICATION

The bit-slice processor design used in the SDFTP was verified via the implementation and testing of a prototype. This simplex prototype processor design included one copy of the microsequencer, control store-pipeline register, and processor array partitions, as shown in Figure 7. Here, the demarcation of the partition boundaries are the dotted lines. The processor has an instruction repertoire of over 100 instructions, listed in Table II. Combining a one microsecond cycle time memory with the simplex processor yields a computer that can execute the flight control programs within the hard deadlines. For typical program mixes, the processing rate is 300,000 - 500,000 operations per second.

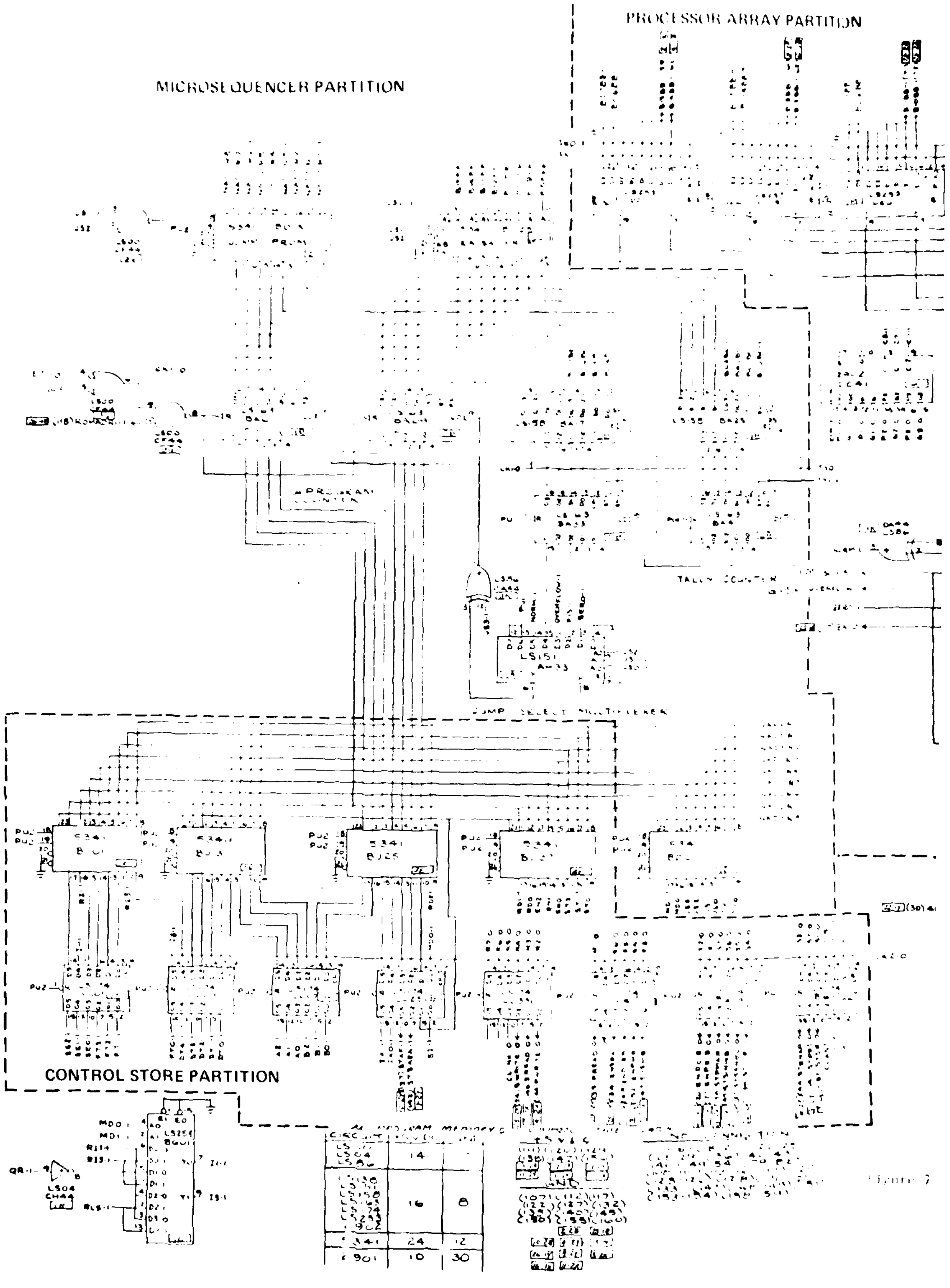

PROCESSOR ARRAY PARTITION
MICROSEQUENCER PARTITION
μPROGRAM COUNTER
TALLY COUNTER
LS151
JUMP SELECT MULTIPLEXER
CONTROL STORE PARTITION
LS04
LS251
+5V & G
GND

Figure 7

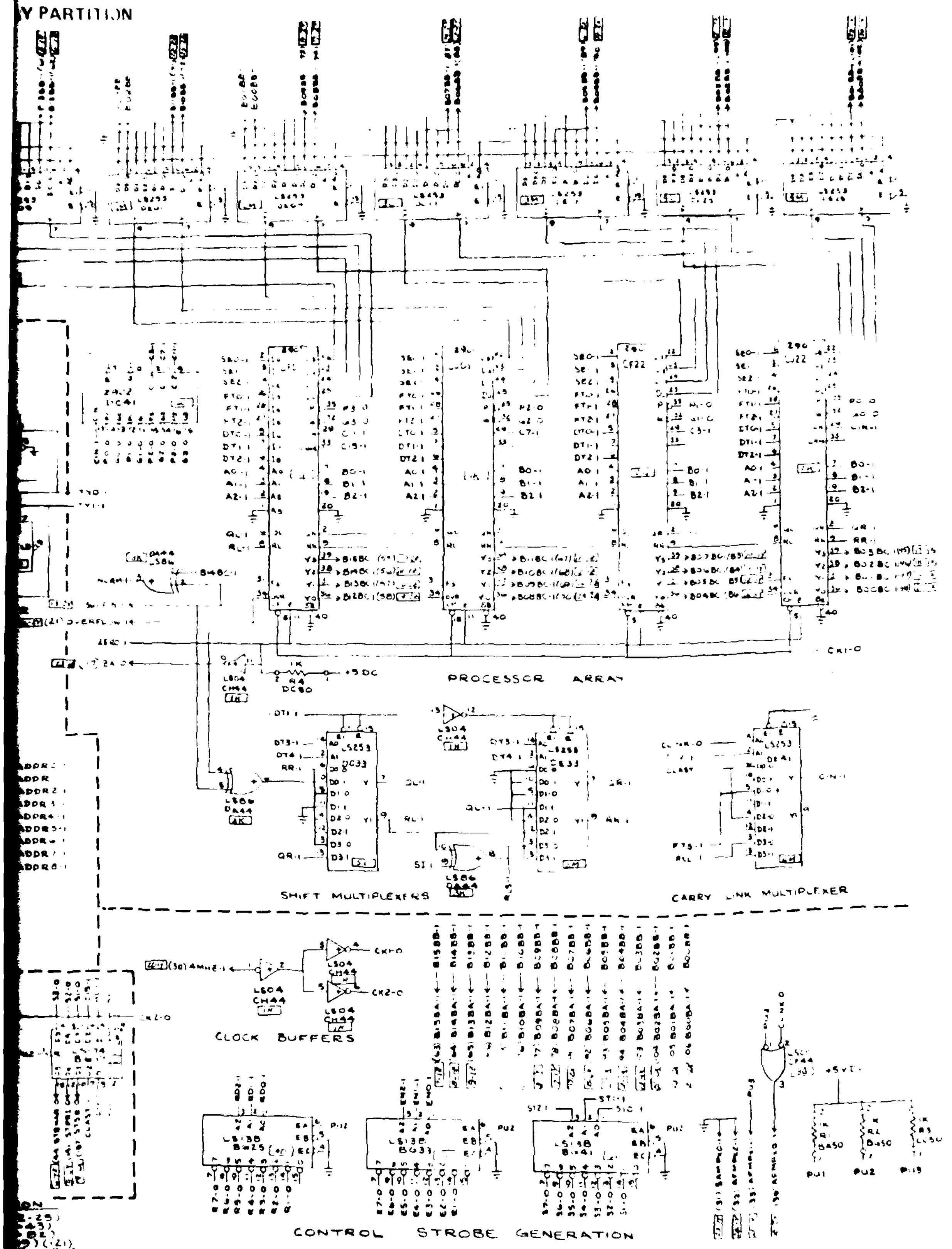


Figure 7. Simplex Prototype Processor Design.

## TABLE II

## ALPHABETICAL LISTING OF MCP-701A INSTRUCTIONS

| Mnemonic | Instruction Description | Instruction Format | Opcode | Execution Time |
|---|---|---|---|---|
| RTN | Return from Subroutine | 4 | 2FXX | 1.0 |
| RINT | Return from Interrupt Routine | 4 | 22XX | 5.0 |
| SBD | Subtract Double from Program Memory | 1 | 54XX | 3.0 |
| SBDS | Subtract Double from Scratchpad Memory | 2 | C0XX | 2.5 |
| SBSP | Set Scratchpad Word Bits Specified | 3 | 74XX | 2.75 |
| SBU | Subtract Program Memory Word from UR | 1 | 52XX | 2.0 |
| SBUS | Subtract Scratchpad Memory Word from UR | 2 | BCXX | 1.5 |
| SIE | Skip If Program Memory Word is Equal to UR | 1 | 60XX | 2.0 - 2.2 |
| SIG | Skip If Program Memory Word is Greater than UR | 1 | 5EXX | 2.0 - 2.2 |
| SIL | Skip If Program Memory Word is Less than UR | 1 | 62XX | 2.0 - 2.2 |
| SILB | Set Indicator (Left Byte) - Immediate | 4 | 1DXX | 1.25 |
| SIRB | Set Indicator (Right Byte) - Immediate | 4 | 1EXX | 1.25 |
| SISE | Skip If Scratchpad Memory Word is Equal to UR | 2 | D8XX | 1.75 - 2.0 |
| SISG | Skip If Scratchpad Memory Word is Greater than UR | 2 | D4XX | 1.75 - 2.0 |
| SISL | Skip If Scratchpad Memory Word is Less than UR | 2 | DCXX | 1.75 - 2.0 |
| SKLB | Skip On Indicator (Left Byte) - Immediate | 4 | 1BXX | 1.25 - 1.5 |
| SKR | Skip On Device Ready | 2 | E4XX | 1.75 - 2.0 |
| SKRB | Skip On Indicator (Right Byte) - Immediate | 4 | 1CXX | 1.25 - 1.5 |
| SKSB | Skip On Scratchpad Word Bits Specified | 3 | 7CXX | 2.0 - 2.25 |
| SLZ | Shift UR Left - Enter Zeros | 4 | 28XX | 1.25 + .25(n) |
| SLZD | Shift Double Left - Enter Zeros | 4 | 29XX | 1.25 + .25(n) |
| SLZX | Shift Double Left By XC - Enter Zeros | 4 | 2EXX | 1.5 + .25(n) |
| SRC | Shift UR Right Circular | 4 | 2AXX | 1.25 + .25(n) |
| SRCD | Shift Double Right Circular | 4 | 2BXX | 1.25 + .25(n) |
| SRS | Shift UR Right-Repeat Sign | 4 | 24XX | 1.25 + .25(n) |
| SRSD | Shift Double Right - Repeat Sign | 4 | 25XX | 1.25 + .25(n) |
| SRSX | Shift Double Right by XC-Repeat Sign | 4 | 2CXX | 1.5 + .25(n) |
| SRZ | Shift UR Right-Enter Zeros | 4 | 26XX | 1.25 + .25(n) |
| SRZD | Shift Double Right - Enter Zeros | 4 | 27XX | 1.25 + .25(n) |

| Mnemonic | Instruction Description | Instruction Format | Opcode | Execution Time |
|---|---|---|---|---|
| STAS | Store XA in Scratchpad Memory | 2 | A0XX | 1.75 |
| STBS | Store XB in Scratchpad Memory | 2 | A4XX | 1.75 |
| STCS | Store XC in Scratchpad Memory | 2 | A8XX | 1.75 |
| STDS | Store Double in Scratchpad Memory | 2 | 9CXX | 3.0 |
| STLS | Store LR in Scratchpad Memory | 2 | 98XX | 1.75 |
| STU | Sote UR in Program Memory | 1 | 4CXX | 2.0 |
| STUS | Store UR in Scratchpad Memory | 2 | 94XX | 1.75 |
| TSU | Transfer SR to UR | 4 | 13XX | 1.25 |
| TUS | Transfer UR to SR | 4 | 14XX | 1.25 |
| XUA | Exchange UR and XA | 4 | 10XX | 1.75 |
| XUB | Exchange UR and XB | 4 | 11XX | 1.75 |
| XUC | Exchange UR and XC | 4 | 12XX | 1.75 |
| XUL | Exchange UR and LR | 4 | 0FXX | 1.75 |
| ZRD | Zero Double (UR and LR) | 4 | 15XX | 1.5 |

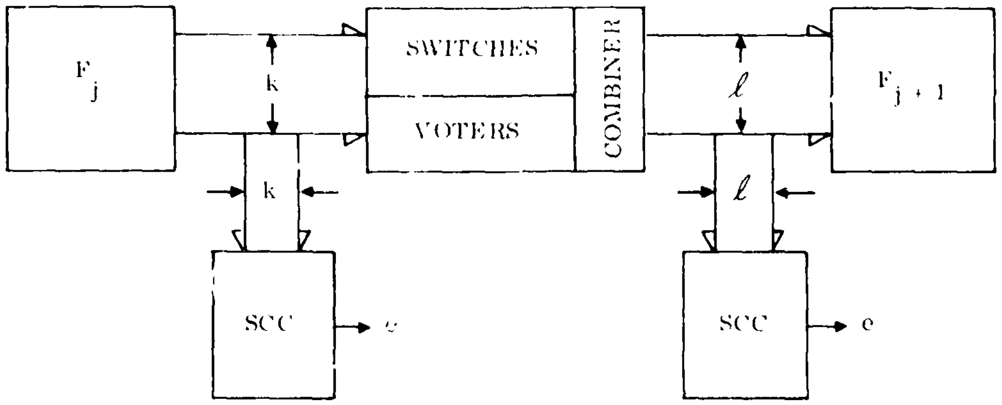
## TABLE II - (Continued)

| Mnemonic | Instruction Description | Instruction Format | Opcode | Execution Time |
|---|---|---|---|---|
| ABSD | Absolute Value of Double Register | 4 | 1AXX | 1.25 - 2.25 |
| ABSU | Absolute Value of UR | 4 | 19XX | 1.25 - 1.75 |
| ADBA | Add to XA (Right Byte) - Immediate | 4 | 0CXX | 1.25 |
| ADBB | Add to XB (Right Byte) - Immediate | 4 | 0DXX | 1.25 |
| ADBC | Add to XC (Right Byte) - Immediate | 4 | 0EXX | 1.25 |
| ADBL | Add to LR (Right Byte) - Immediate | 4 | 0BXX | 1.25 |
| ADBU | Add to UR (Right Byte) - Immediate | 4 | 0AXX | 1.25 |
| ADD | Add Double from Program Memory | 1 | 50XX | 3.0 |
| ADDS | Add Double from Scratchpad Memory | 2 | B0XX | 2.5 |
| ADMS | Add Double to Scratchpad Memory | 2 | B8XX | 4.25 |
| ADU | Add to UR from Program Memory | 1 | 4EXX | 2.0 |
| ADUS | Add to UR from Scratchpad Memory | 2 | ACXX | 1.5 |
| AMS | Add UR to Scratchpad Memory | 2 | B4XX | 2.5 |
| CBSP | Clear Scratchpad Word Bits Specified | 3 | 78XX | 2.75 |
| CILB | Clear Indicator (Left Byte)-Immediate | 4 | 1FXX | 1.25 |
| CIRB | Clear Indicator (Right Byte) - Immediate | 4 | 20XX | 1.25 |
| CLR | Clear Device Controller | 2 | E8XX | 1.25 |
| CLRI | Clear Interrupt Specified | 4 | 23XX | 1.25 |
| CPLD | Complement Double Register | 4 | 18XX | 1.75 - 2.0 |
| CPLU | Complement UR | 4 | 17XX | 1.5 |
| DIV | Divide Double by Program Memory Word | 1 | 58XX | 10.75 |
| DIVS | Divide Double by Scratchpad Memory Word | 2 | C8XX | 10.5 |
| DSSZ | Decrement and Skip if Scratchpad Word is Zero | 2 | E0XX | 2.5 - 2.75 |
| ENBL | Enable Device to Interrupt | 2 | ECXX | 1.25 |
| INHB | Inhibit Device from Interrupting | 2 | F0XX | 1.25 |
| INV | Invert UR | 4 | 16XX | 1.25 |
| JINT | Jump to Service Interrupt | 4 | 21XX | 8.0 |
| JMP | Jump Unconditional | 1 | 64XX | 1.5 |
| JMPI | Jump Unconditional, Indirect | 1 | 66XX | 2.0 |
| JMS | Jump to Subroutine | 1 | 68XX | 1.5 |
| JMSI | Jump to Subroutine, Indirect | 1 | 6AXX | 2.0 |

| Mnemonic | Instruction Description | Instruction Format | Opcode | Execution Time |
|---|---|---|---|---|
| JSNS | Jump After Interrupt Sense | 2 | F4XX | 2.75 |
| LALB | Load XA (Left Byte) - Immediate | 4 | 02XX | 1.25 |
| LARB | Load XA (Right Byte) - Immediate | 4 | 07XX | 1.25 |
| LBLB | Load XB (Left Byte) - Immediate | 4 | 03XX | 1.25 |
| LBRB | Load XB (Right Byte) - Immediate | 4 | 08XX | 1.25 |
| LCLB | Load XC (Left Byte) - Immediate | 4 | 04XX | 1.25 |
| LCRB | Load XC (Right Byte) - Immediate | 4 | 09XX | 1.25 |
| LDA | Load XA from Program Memory | 1 | 46XX | 2.0 |
| LDAS | Load XA from Scratchpad Memory | 2 | 88XX | 1.5 |
| LDB | Load XB from Program Memory | 1 | 48XX | 2.0 |
| LDBS | Load XB from Scratchpad Memory | 2 | 8CXX | 1.5 |
| LDC | Load XC from Program Memory | 1 | 4AXX | 2.0 |
| LDCS | Load XC from Scratchpad Memory | 2 | 90XX | 1.5 |
| LDD | Load Double from Program Memory | 1 | 44XX | 3.0 |
| LDDS | Load Double from Scratchpad Memory | 2 | 84XX | 2.5 |
| LDL | Load LR from Program Memory | 1 | 42XX | 2.0 |
| LDLS | Load LR from Scratchpad Memory | 2 | 80XX | 1.5 |
| LDU | Load UR from Program Memory | 1 | 40XX | 2.0 |
| LDUS | Load UR from Scratchpad Memory | 2 | 7CXX | 1.5 |
| LLLB | Load LR (Left Byte) - Immediate | 4 | 01XX | 1.25 |
| LLRB | Load LR (Right Byte) - Immediate | 4 | 06XX | 1.25 |
| LULB | Load UR (Left Byte) - Immediate | 4 | 00XX | 1.25 |
| LURB | Load UR (Right Byte) - Immediate | 4 | 05XX | 1.25 |
| MPY | Multiply UR by Program Memory Word | 1 | 56XX | 6.0 |
| MPYS | Multiply UR by Scratchpad Memory Word | 2 | C4XX | 5.75 |
| NDU | And to UR from Program Memory | 1 | 5AXX | 2.0 |
| NDUS | And to UR from Scratchpad Memory | 2 | CCXX | 1.5 |
| NRM | Normalize Double | 4 | 2EXX | 2.0 + .25(n) |
| ORU | Or to UR from Program Memory | 1 | 5CXX | 2.0 |
| ORUS | OR to UR from Scratchpad Memory | 2 | D0XX | 1.5 |

## C. INTERFACE MODULE DESIGN

In between the triplicated partitions of the bit-slice processor are the partition interface modules as shown in Figure 8. These modules are a combination of partition output SCC's, Voter-Switches, and partition input SCC's. The function of the SCC's is to produce error patterns so that the bit-slice processors can be properly reconfigured to switch out a failed partition or part of an interface module. Using the Voter-Switch devices, the faulted device can be isolated while maintaining as high a level of redundancy as possible throughout the design. The usage of these devices in the interfaces is described next. For more detailed information see Section III-G, Custom Devices.



VOTER & SWITCH

- ~ 350 GATES
- 64 PIN

Figure 8 Voter and Switch and Self-Checking Checkers

### 1. Self-Checking Checker Without Mask

The S.C.C. detects errors in its inputs as well as any faults in the checker itself. In a self-checking circuit such as the S.C.C. the inputs and outputs are encoded so that any assumed fault within the circuit, or any non-code input, produces a non-code output for at least one of the normally occurring inputs. The validity of the output code words is checked by a check circuit that produces valid/invalid indications. The check circuit is also designed so that it is self-checking and produces the same "invalid" indication for a fault in the check circuit as for non-code inputs. So called "totally self-checking" (TSC) circuits, which are one of the classes of self-checking circuits, were proposed by D.A. Anderson (1). This class of circuit has the property that an assumed fault never causes an erroneous code output in addition to the attributes already cited. (For a rigorous definition of TSCC see reference 4.)

In this flight-control application, the fault model selected is the so-called "stuck-at model," where hardware failures in a circuit are modeled as some logic gate input or logic gate output lines stuck-at-1 (S-a-1) or stuck at-0 (S-a-0). Faults are said to occur when one or more lines become S-a-1 or S-a-0. Thus, when a single line is stuck, a single fault is said to have occurred. Multiple faults, where more than one line is stuck, may occur. If one or more lines become stuck at the same logic value, i.e. 1 or 0, a unidirectional fault is said to have occurred.

Once the fault model has been elected, the checker code can be determined. The choice of the code can strongly affect the property of the checkers and functional blocks. Constant weight and unordered codes have been suggested (2,3) for the design of totally self-checking circuits. These codes are used because the structure of the functional blocks and the assumed faults always lead to unidirectional errors (4). Unordered codes have the property that they can detect any unidirectional error. A consequence of the use of unordered codes is that totally self-checking checkers must be built to check them.

The code selected for the bit-slice processor partitions is the triplication code, which is then partitioned into a trio of duplication codes for input to the S.C.C. The code selected for the S.C.C. is the dual-rail code. Thus, within the S.C.C., one of the signals of each of the pairs of signals of the triplicated partition signals is inverted. In the S.C.C. device up to 16 triple inputs can be checked. Within the S.C.C.'s, each of the three pairs of signals is checked by a self-checking tree consisting of 4-out-of-8 T.S.C's as shown in Figure 9. The output of each tree is a dual-rail signal, in which an error is indicated by a *0·0* or *1·1* combination. *These three tree*

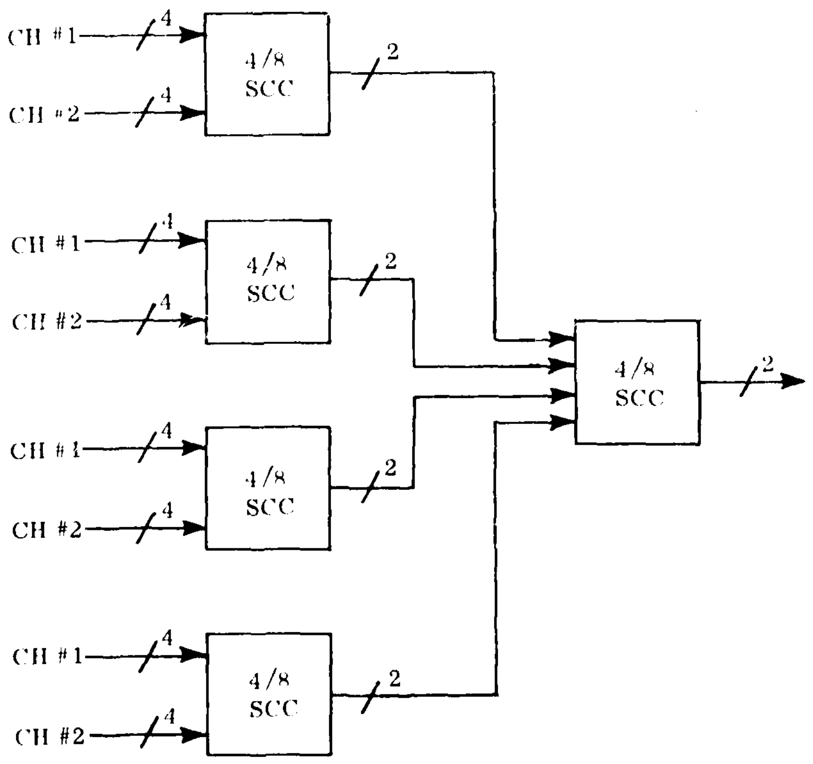


Figure 9. Totally Self-Checking Checker (TSC)

outputs are combined with snapshots of the three current input signals, and the three inputs from the previous microcycle. These signals are saved in three separate scanning registers as shown in Figure 10. These three snapshots are identical except for the error field. The 6-bit error vector of each snapshot is rotated with respect to the other two snapshot error fields so that the dual-rail output of each of the three T.S.C. trees is available externally. This permits a check on all input signals to the S.C.C. The field definitions of the scanning register are shown in Figure 11.

By clocking the inputs and outputs of the three T.S.C.'s into the three snapshot registers every microcycle, the current status of the interface is captured. The current input (i) is shifted down to the previous microcycle input field (i-1) on the occurrence of the next clock cycle. Thus, where an error is detected by one of the S.C.C.'s it causes the RRU Maskable S.C.C. to indicate an error, which stops the processor clocks that clock the snapshots into the snapshot register. Thus, the cause of the error and its location are available for diagnosis by the RRU. Each snapshot register can be clocked by a different processor clock if desired.

Similarly, each snapshot register can be examined by clocking out the contents. In the SDFTP, the RRU shifts out the snapshot by supplying a shift clock to each register. Hence, the snapshots can be read independently by the RRU computers. This occurs after the error outputs have been processed to turn off the processor clocks, and the RRU computers have been alerted that an error has been detected by being interrupted. The contents of the scanning register can be repeated since the information is recirculated as it is read out, if an error is detected during the first read-out. Under some conditions only part of the snapshot, such as the error vector, may be read out.

The outputs of the S.C.C. are connected to the RRU Maskable S.C.C. trees and to the Input Buffers of the RRU computer. Each of these S.C.C. trees combines all of the S.C.C. signals of the devices, located at at the interface, into a single dual-rail signal, which interrupts its associated RRU computer and bit-slice processor clock. The other connection of the interface S.C.C. involves only one third of all of the S.C.C. outputs. Each RRU computer is connected to only one of the three scanning registers of each of the S.C.C.'s. By connecting the second and third scanning registers of each interface S.C.C. to different RRU computers, all three computers have complete but independent access to the error vectors and S.C.C. inputs. By shifting these scanning registers out under the shift clock control, as much of the snapshot can be obtained as needed for diagnosis.

Since each S.C.C. without mask can handle 16 or less inputs, all of the four interfaces require just two devices, with the exception of the control store-processor array interface. This interface requires six devices since there are less than 48 signals that must be protected at the partition output and inputs. Thus a total of 12 devices are needed per SDFTP.

### 2. Voter-Switch

After a detected error has been diagnosed the connections between the bit-slice processor may need to be changed, and the status of the Voter-Switch devices at each interface, which permit this automatic reconfiguration under the control of an RRU computer, may be altered. Each Voter-Switch device incorporates a triplicated set of circuits for nine triplicated signals. Each circuit consists of a three input voter and set of three switches. Both

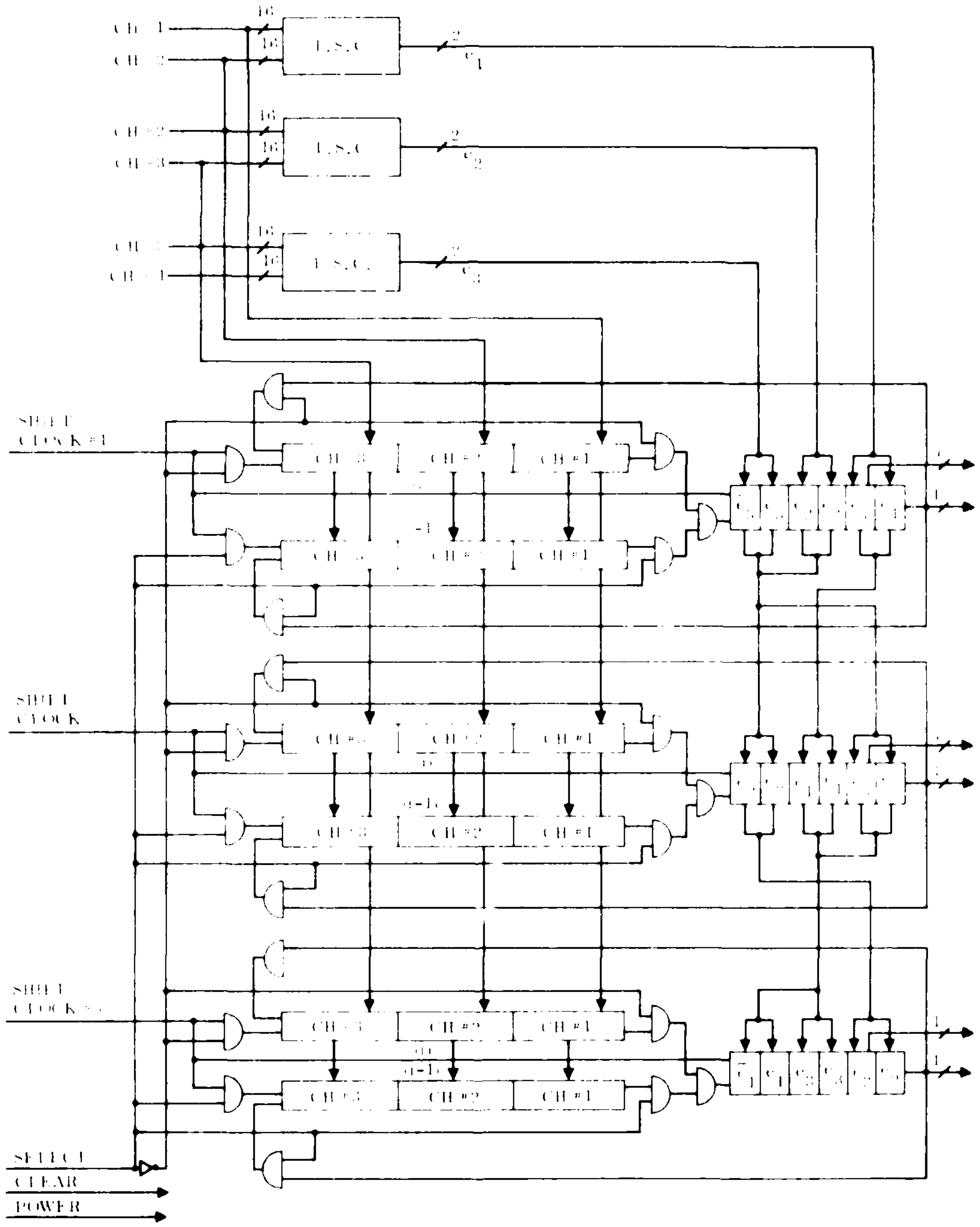


Figure 10. S.C.C. Without Mask

SCANNING REGISTER #1

| | | | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|
| INPUT 3 (i-1) | INPUT 2 (i-1) | INPUT 1 (i-1) | INPUT 3 (i) | INPUT 2 (i) | INPUT 1 (i) | $\bar{e}_3$ | $e_3$ | $\bar{e}_2$ | $e_2$ | $\bar{e}_1$ | $e_1$ |

SCANNING REGISTER #2

| | | | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|
| INPUT 3 (i-1) | INPUT 2 (i-1) | INPUT 1 (i-1) | INPUT 3 (i) | INPUT 2 (i) | INPUT 1 (i) | $\bar{e}_2$ | $e_2$ | $\bar{e}_1$ | $e_1$ | $\bar{e}_3$ | $e_3$ |

SCANNING REGISTER #3

| | | | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|
| INPUT 3 (i-1) | INPUT 2 (i-1) | INPUT 1 (i-1) | INPUT 3 (i) | INPUT 2 (i) | INPUT 1 (i) | $\bar{e}_1$ | $e_1$ | $\bar{e}_3$ | $e_3$ | $\bar{e}_2$ | $e_2$ |

S.C.C. WITHOUT MASK

Figure 11. Scanning Register #1

the voter and the switches receive the same triplicated signals as shown in Figure 12. The voter computes the majority function of the three inputs while the switch selects one of the three inputs for transmission to the output. Only one or the other is connected to the output at any one time and the connection is controlled by a command register that can be set externally. In the case of the SDFTP, the RRU computer transmits the command from its output buffer to the Voter-Switch in serial form. The information is then clocked into the command register by the command clock, which is also supplied by the RRU computer. Each of the three sets of nine input signals is controlled by a separate command register. For the SDFTP application each command register is controlled by a separate RRU computer and can be loaded independently. In Figure 12 a single bit slice is shown to the right of the dotted line and the three command registers are shown to the left of the dotted line. Each command register controls one of the three voter-switch circuits.

The voter-or-switch (VOS) bit of the command dictates which device, voter or switch, will provide the output. When VOS is a "1", the voters are selected while "0" selects the switches. The command bits C1, C2, C3 determine which of the switches is connected to the input. A "1" in any of these three bit positions selects the corresponding switch; i.e. C1=1, the top of the three switches. Only one switch should be used to maintain the independence of the channels.

A total of 10 of these devices is required to provide this function for all of the SDFTP interfaces. The processor array-microsequencer and microsequencer control store interface each require one Voter-Switch device. The processor array-memory interface requires two devices while the control store-processor array requires six.

## D. RRU OPERATION

### 1. Architecture Review

First, the architecture of the error reporting system will be reviewed because a good understanding of the various parts of the system and the way they interrelate will make it easier to understand the overall operation of the RRU.

Figure 13 is a simplified overview of the processor and the SCC's. Each of these components is discussed in detail in Section III G. The basic processor partitions, including the microsequencer ($\mu$s) control store (CS) and processor array (PA) are shown separated by voter/switches labeled V/S. The control store and processor array require more than one voter/switch to handle all their output leads. The three processor channels are illustrated with a three-dimensional effect.

Errors are reported by SCC's. A net of three TSC's labeled "out" is connected to the output of each processor partition. Likewise, the TSC's labeled "In" are connected to the input of each processor partition. These circuits check for errors at the input and output of each processor partition and provide sufficient information to locate errors in either the partitions or the switches. The error signals from these partition SCC's are merged

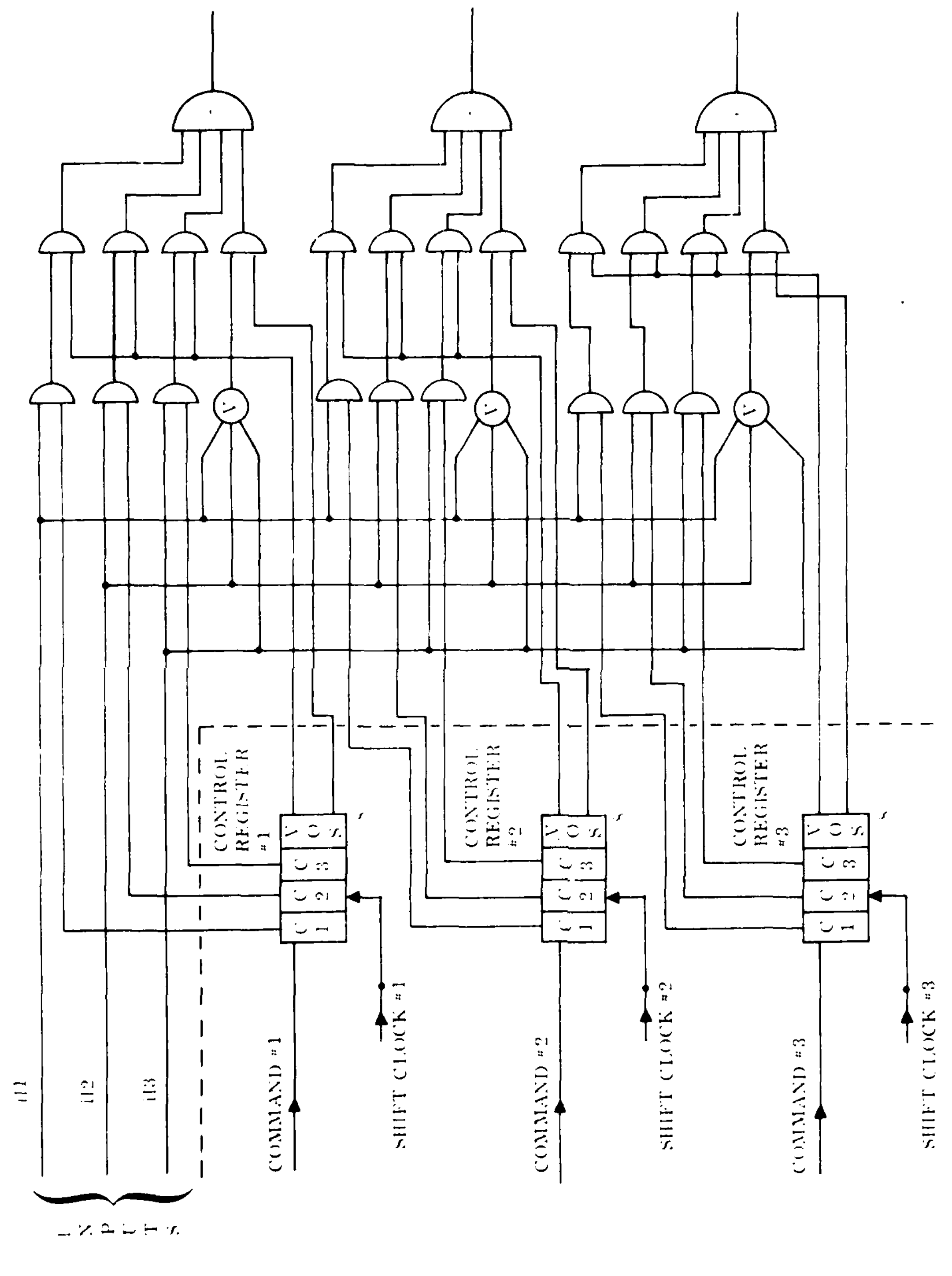


Figure 12. Triplicated Voter and Switch

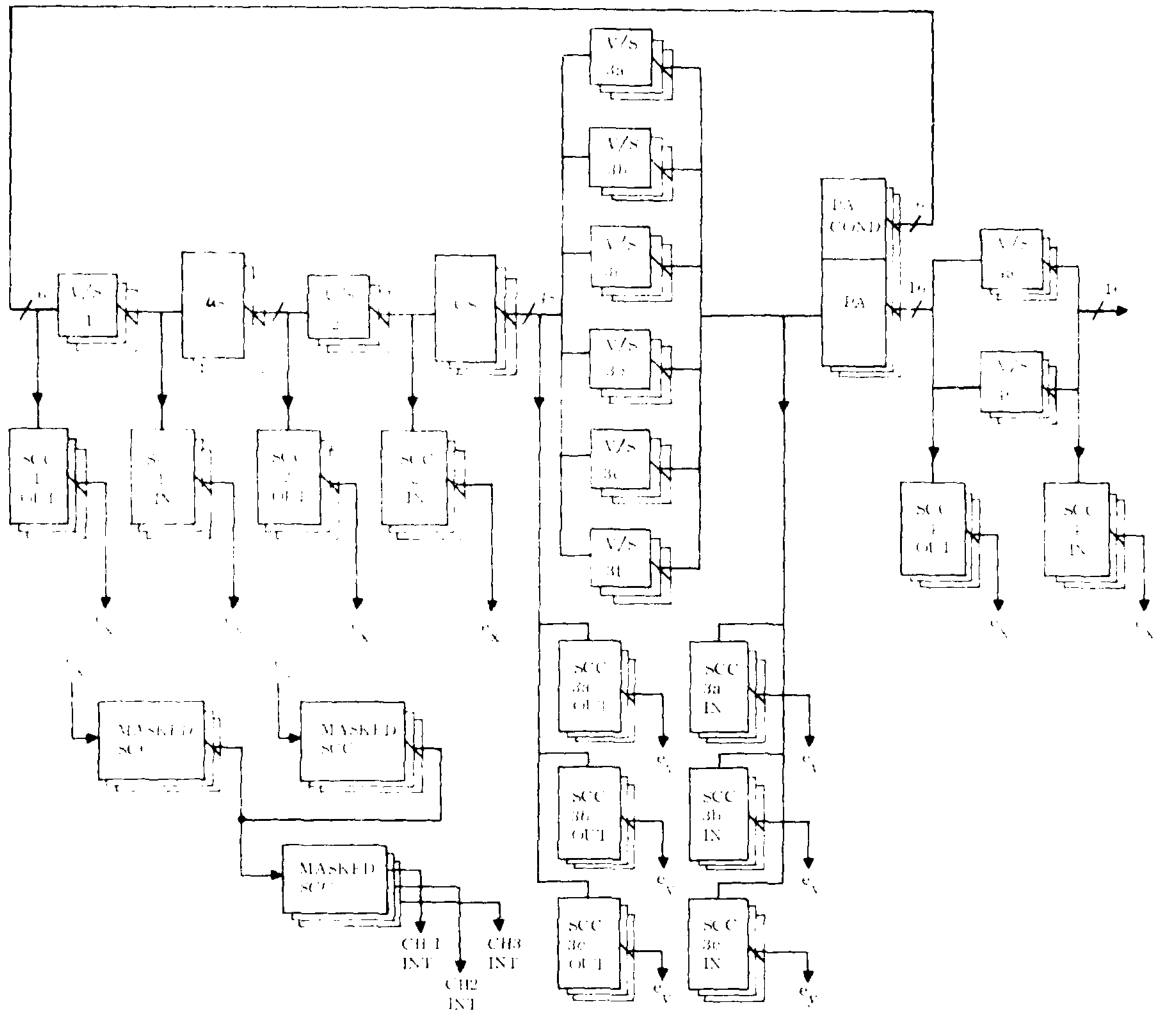


Figure 13. Processor System Overview

together via special masked SCC's, which provide interrupt signals for each of three RRU's (reconfiguration recovery units). The interrupt signal is generated whenever any unmasked error is obtained from the system.

The interrupt signals cause the RRU's shown in Figure 14 to go to an error processing routine, which determines the cause of the error and issues the appropriate commands to switch data signals around the fault in an optimum manner.

The RRU processors are implemented with monolithic microprocessors with appropriate memory and I/O circuits. They are described more fully in Section III E . The RRU processors are operated independently for the most part. They are loosely synchronized in that they can communicate with each other and the maintenance panel by handshake signals. They also perform a type of synchronized control when turning on the processor clocks such that the processors are locked together. The interrupts that initiate the error response processing are also provided in triplicate and each RRU processor can select any one of the three.

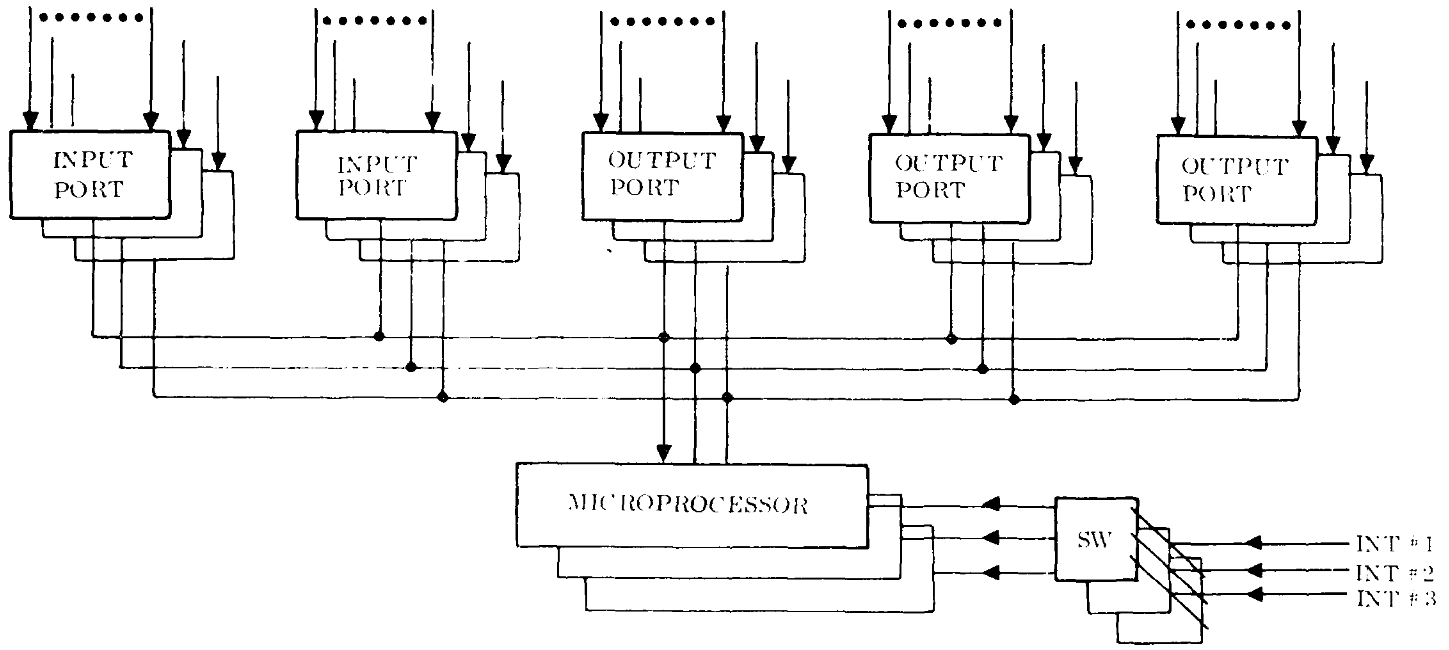


Figure 14. Reconfiguration Recovery Unit Overview

It is important to understand, in some detail both the manner in which the processor system can be reconfigured with the voter switches and the way in which the error signals are derived. The RRU system is the link from the error signals to the voter switch controls. The number 2 voter switch and the 2 OUT and 2 IN SCC's are shown in functional detail in Figure 15. Each voter switch receives all three channel signals coming from the microsequencer. The voter switch outputs can be derived by a vote of all three inputs or they can be switched directly to any one of the three inputs. Initially, the voter is used so that a single microsequencer failure is masked by the voter. Should two microsequencers fail, then each switch is set to use the one remaining microsequencer as an input. Each of the following control stores (CS) would thus receive a correct input.

The number 2 OUT SCC is connected to the outputs of the three microsequencers as shown in Figure 15. Likewise, the number 2 IN SCC is connected to the inputs of the three control stores. The $e_1$ error signal is derived by comparing channels 1 and 2; and $e_2$ is derived from channels 2 and 3 and $e_3$ from channels 3 and 1. The number 2 SCC IN error signals, $e_1'$, $e_2'$ and $e_3'$, are derived in a similar fashion from the 1', 2' and 3' signals.

The interpretation of these error signals is as shown in Figure 16. Failures associated with each error pattern are listed here. The interpretations of the first fault is relatively straightforward as indicated by the entries in the column titled Single Faults. The only entry that might seem confusing is the interrupt error. This is detected when an interrupt yields no error signals in any of the SCC's. Since the SCC outputs are only examined after an interrupt has occurred, the only conclusion that can be reached is that the interrupt circuitry has failed. The RRU circumvents the problem by switching its interrupt input to one of the other two interrupt signals.

Consider, now, the problem when a second fault is sensed in a single output SCC. There are several possibilities for each of the double error patterns. The RRU's now must resort to either running processor diagnostics or performing a series of sequential tests to determine the cause of the second fault.

The operation of the system, where two faults occur in the same SCC partition is best described by an example. Assume that the first fault is the failure of the channel 1 microsequencer. The second fault is then assumed to be the failure of the channel 2 microsequencer. This can result in a situation where all three error signals indicate errors. As seen in Figure 16 even though channel 1 had been identified as the first error it may be impossible to determine whether the second fault is due to a failure in channel 2 or 3. Thus, a processor diagnostic is run to determine which channel is at fault.

Another example is the failure of SCC 1 followed by the failure of microsequencer 1, having an error pattern consisting of $e_1$ and $e_3$. However, it is not clear whether the second fault is due to a SCC 3 or a channel 1 failure, since either of these failures in combination with the original SCC 1 failure yields the same error pattern. Here, a sequential test is used.

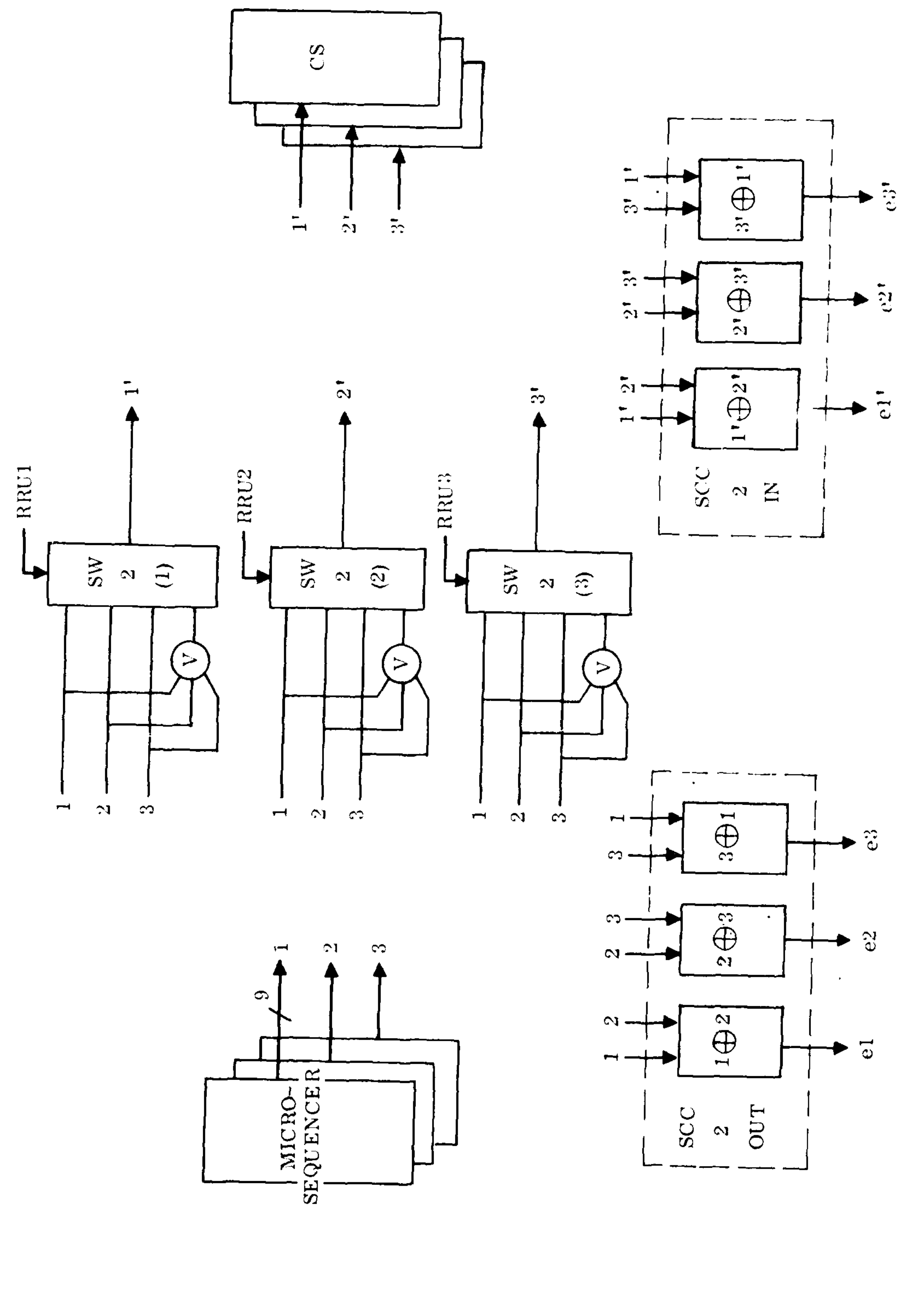


Figure 15. Functional Detail of Voter/Switches and SCC's

| SINGLE FAULTS | DOUBLE FAULTS | $e_1$ $1 \oplus 2$ | $e_2$ $2 \oplus 3$ | $e_3$ $3 \oplus 1$ |
|---|---|---|---|---|
| INT. | | | | |
| SCC1 | INT. + SCC1 | X | | |
| SCC2 | INT. + SCC2 | | X | |
| SCC3 | INT. + SCC3 | | | X |
| CH1 | INT. + CH1, SCC1 + SCC3, SCC1 + CH1, SCC3 + CH1 | X | | X |
| CH2 | INT. + CH2, SCC1 + SCC2, SCC1 + CH2, SCC2 + CH2 | X | X | |
| CH3 | INT. + CH3, SCC2 + SCC3, SCC2 + CH3, SCC3 + CH3 | | X | X |
| | CH1 + CH2, CH2 + CH3, CH1 + CH3<br>SCC1 + CH3, SCC2 + CH1, SCC3 + CH2 | X | X | X |

Figure 16. Interpretation of SCC Error Signals

The SCC 3 error is masked and the voter/switch is switched to the channel 1 input and the main processor is restarted. If there is no error interrupt then the second failure was, in actuality, an SCC 3 error. But if the failure was microsequencer 1, the input SCC following the voter/switch will indicate a channel 1 error, since channel 1 is being compared to the other voter outputs at this point. Upon detecting a channel 1 error the swiich is either thrown back to the voter position or to channel 2 or 3.

It should be emphasized that these double-fault problems occur only when two faults are reported by a single SCC. Two faults each occurring in different SCC's are processed as successive single faults.

Two simultaneous faults in a single SCC are recognized, and diagnostics are used to determine any failed channels. The remaining faults, if any, are assumed to be SCC errors.

The error interpretation for input SCC's is similar, with the exception that the channel errors are now caused by voter/switches. A failure here results in switching to a good function input. This causes the error pattern to revert to the no-error case, and a second error of this type will cause a single fault pattern and be easily detected and identified.

Maps of the system errors for three system states are shown in Figure 17. The map is laid out in the same order as the system components shown in Figure 13. The blank boxes represent the SCC's and the other boxes are the other elements in the system as labeled. The first map illustrates the no-error conditions. The second map indicates $e_1$ and $e_3$ errors in SCC 2 OUT, which is a channel 1 $\mu$s fault. The third map adds a channel 2 voter/switch fault to the previous fault, resulting in $e_1$ and $e_3$ errors in SCC 4 IN.

The voter/switches each switch nine bits of data, whereas the SCC's report errors in 16-bit slices. This results in some overlap of switchable slices between SCC's for the design described. This situation is depicted in Figure 18. An example is SCC 3b, which monitors parts of slices 3b, 3c and 3d. There is no way to determine which 9-bit slice caused the error by examining the error signals alone. Currently, the system assumes that all three slices are bad and acts accordingly. It would be possible to distinguish between these slices if the actual data are read from the SCC registers. However, this would take considerable time and there seems to be no advantage to isolating the fault to one slice for reconfiguration purposes. The exact locations of the fault for maintenance purposes would be accomplished by the RRU self-test programs.

## 2. Error Response Operations

The initial part of the RRU operational flow diagram, which performs the error interpretation and reconfigures the system via the switches, is shown in Figure 19. When the main processors are running, the RRU's are either idle or running self test, waiting for an error interrupt. When the interrupt occurs, the errors are read by the RRU processors via their I/O ports. The e signals from all the SCC's are available to each RRU. These are the same signals that are merged together with the masked SCC's to provide the interrupt signals.

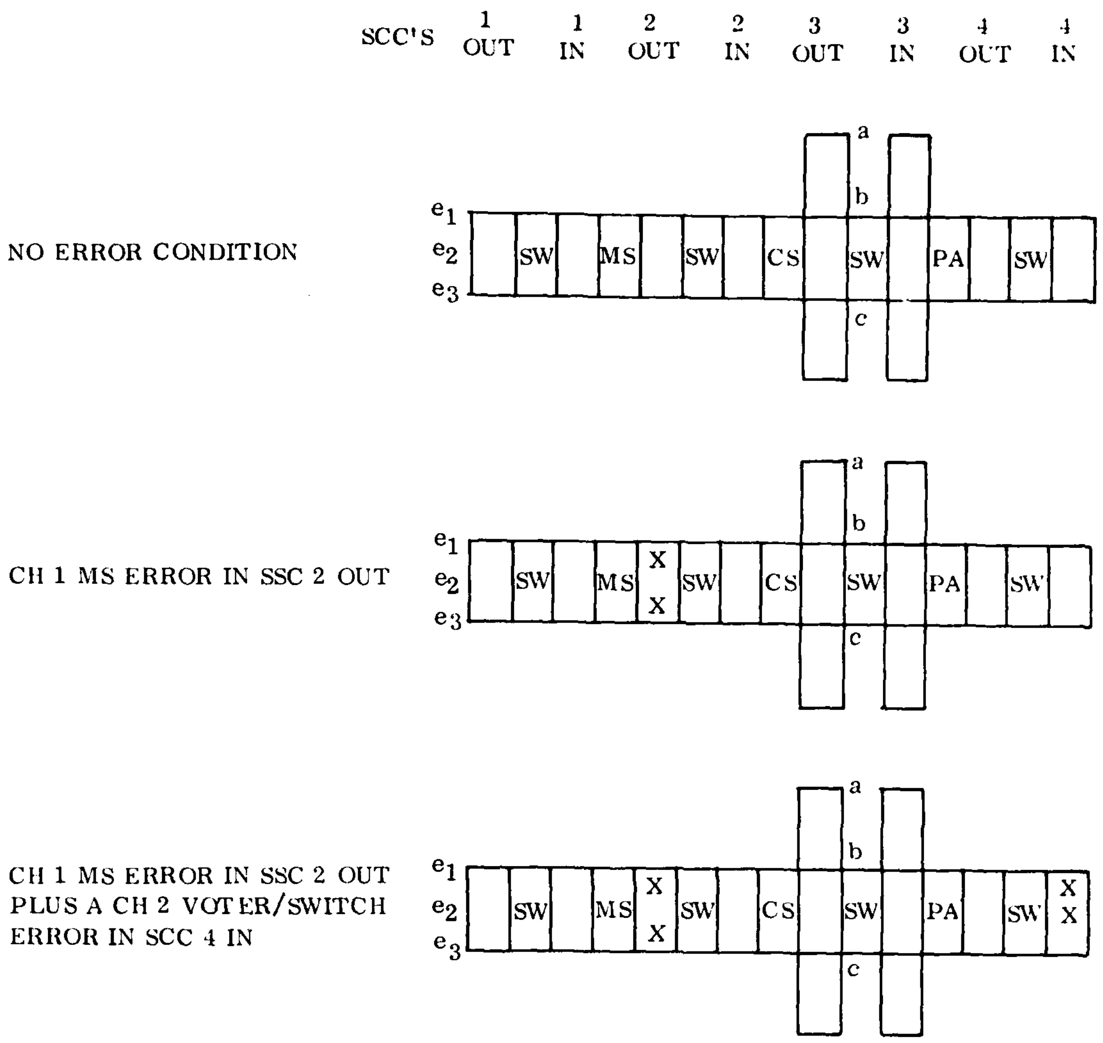

SCC'S
1 OUT
1 IN
2 OUT
2 IN
3 OUT
3 IN
4 OUT
4 IN
NO ERROR CONDITION
$e_1$
$e_2$
$e_3$
a
b
c
SW
MS
SW
CS
SW
PA
SW
CH 1 MS ERROR IN SSC 2 OUT
X
X
CH 1 MS ERROR IN SSC 2 OUT PLUS A CH 2 VOTER/SWITCH ERROR IN SCC 4 IN

Figure 17. System Error Maps

| | | Switch # | Bits | SCC# 1<br>Bits 1-16 | SCC# 2<br>Bits 1-16 | SCC# 3a<br>Bits 1-16 | SCC# 3b<br>Bits 17-32 | SCC# 3c<br>Bits 33-48 | SCC# 4<br>Bits 1-16 |
|---|---|---|---|---|---|---|---|---|---|
| PA | COND | 1 | 1-6 | X | | | | | |
| | MS | 2 | 1-9 | | X | | | | |
| | CS | 3a | 1-9 | | | X | | | |
| | | 3b | 10-18 | | | X | X | | |
| | | 3c | 19-27 | | | | X | | |
| | | 3d | 28-36 | | | | X | X | |
| | | 3e | 37-45 | | | | | X | |
| | | 3f | 46-48 | | | | | X | |
| | PA | 4a | 1-9 | | | | | | X |
| | | 4b | 10-16 | | | | | | X |

Figure 18. Switch Fault Error Reporting

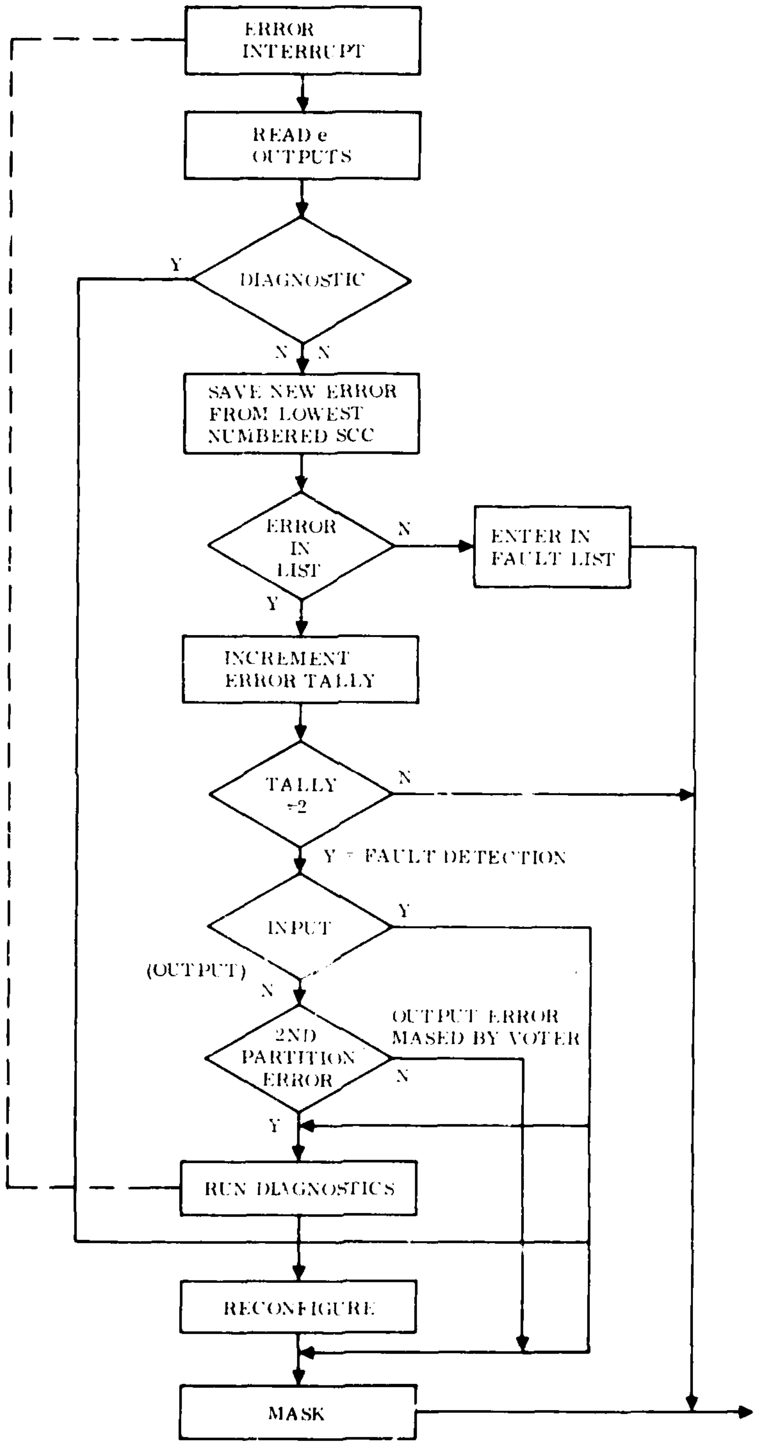


Figure 19. First Part of RRU Operational Flow Diagram

The order of the e signals on the RRU ports for one of the RRU computers is depicted in Figure 20. All eight bits of port 1 and four bits of port 2 are used. The signals are obtained by shifting them out of the SCC registers, starting with $e_1$ and ending with $\bar{e}_3$. The order shown in Figure 20 is the order for the channel-1 RRU processor. The channel-2 processor receives the $e_2$ signals followed by $e_3$ and then $e_1$. The channel-3 processor receives $e_3$ first followed by $e_1$ and $\bar{e}_2$, in that order. The orders cannot be the same because the same output pins must supply the masked SCC's with all three error signals in parallel. The RRU must send a clock to the SCC's each time it wants a new set of error signals. The signals on one port are all read at the same time. An error is recognized if any $e_i$, $\bar{e}_i$ pair are the same. The normal flow then continues to the box in Figure 19 labeled "save new error from lowest numbered SCC". The logic must reject all error signals from previous faults that have been processed. These errors remain in the system and may be read each time an interrupt is generated. (They are prevented from causing interrupts by masking in the merging SCC's.) The logic then selects the error bit combinations from the lowest numbered SCC. This ensures that only one new error will be processed at a time.

The next step in the flow diagram of Figure 19 is to determine whether this error report already exists in a fault list of previous detected faults. If it is not in the list the error is entered in the list and processing is resumed. If this error has already occurred once it will be in the list. A particular error is not designated a fault until the second time it occurs. This prevents transient errors from reconfiguring the system needlessly.

Once a fault is detected it is tested to see if it comes from an input or output SCC. If it is an input error some processing is always needed. It may involve running a diagnostic or just reconfiguring. If the error is from an output SCC it is checked to determine whether it is the first or second fault in the SCC partition. A single partition fault is generally masked by the following voter and no reconfiguration is needed. Second faults are generally more complicated and require running some sort of diagnostic. The exact logic path followed by a given fault is determined by the particular combination of faults. There are several special cases that do not follow this diagram exactly.

A flow diagram showing the processor diagnostic procedure in more detail is given in Figure 21. The diagnostic is selected by the RRU sending a diagnostic starting address to the processor. The RRU then sets the interrupt circuits into a mode that only allows interrupts to be generated at the end of the diagnostic. If this were not done the processors would be stopped as soon as the first error appeared in the diagnostic run. The RRU processors, however, only know the correct results obtained at the end of the diagnostic. Therefore, the diagnostic must run to completion to permit the RRU to determine which channel has failed. The diagnostic is started by enabling the main processor clocks; it then runs until the results at the end of a diagnostic are in error at which time an interrupt is generated. Flags are tested to determine which routines should be run to interpret the test results.

The major steps in the reconfiguration and masking processes are shown in Figure 22. The first step is to generate the proper code word to enable the clock signals needed to change the voter switch state. The

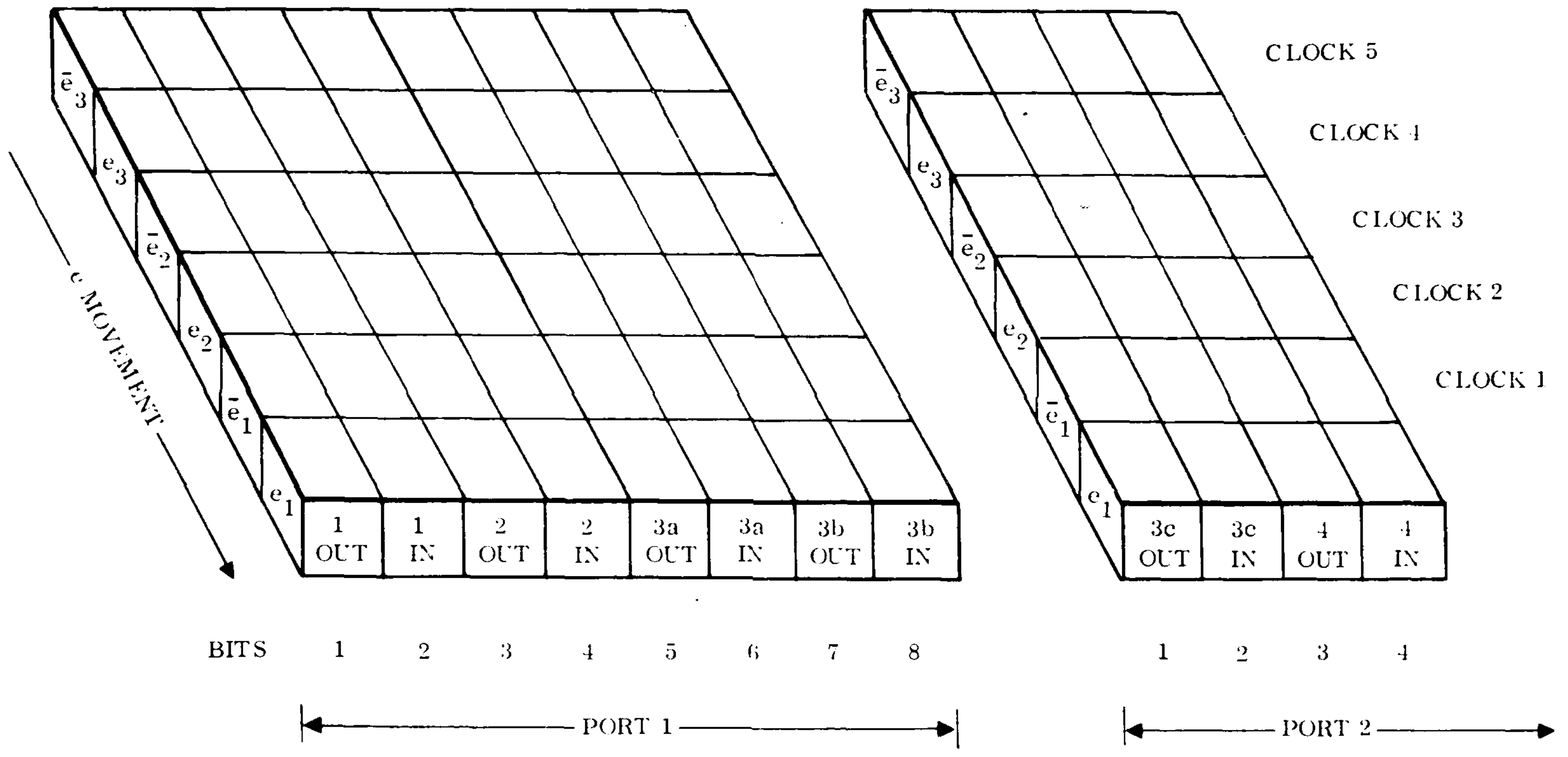


Figure 20. Error Signal Order on RRU I/O Ports

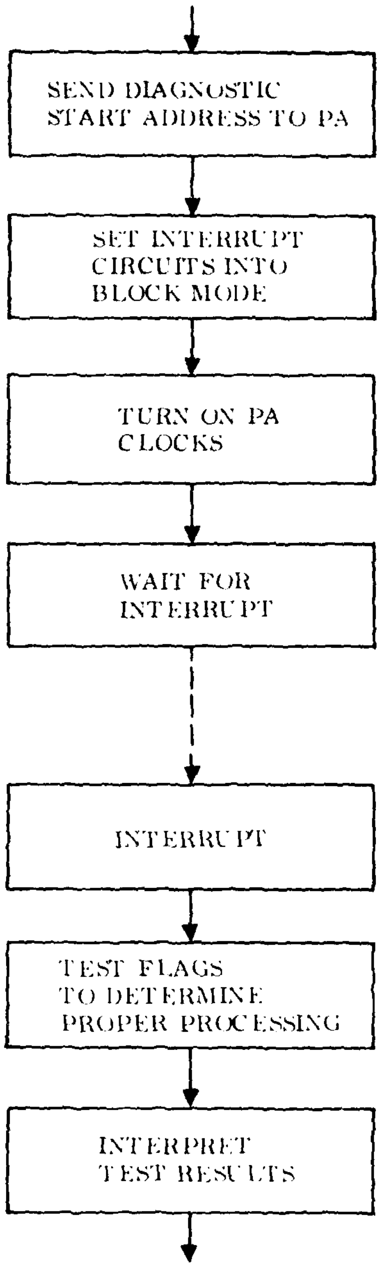
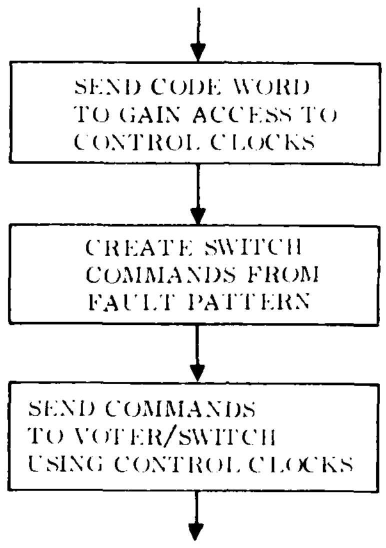

SEND DIAGNOSTIC START ADDRESS TO PA
SET INTERRUPT CIRCUITS INTO BLOCK MODE
TURN ON PA CLOCKS
WAIT FOR INTERRUPT
INTERRUPT
TEST FLAGS TO DETERMINE PROPER PROCESSING
INTERPRET TEST RESULTS

Figure 21. Diagnostic Flow Diagram


SEND CODE WORD TO GAIN ACCESS TO CONTROL CLOCKS
CREATE SWITCH COMMANDS FROM FAULT PATTERN
SEND COMMANDS TO VOTER/SWITCH USING CONTROL CLOCKS

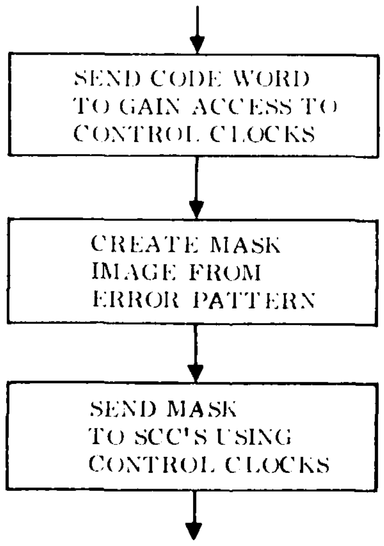

SEND CODE WORD TO GAIN ACCESS TO CONTROL CLOCKS
CREATE MASK IMAGE FROM ERROR PATTERN
SEND MASK TO SCC'S USING CONTROL CLOCKS

Figure 22. Reconfiguration and Masking Flow Diagrams

requirement of generating a code to turn on the clocks makes it quite unlikely that a runaway RRU could accidently gain access to the switch controls. The next step in the reconfiguration is to create a switch command message from the fault information. This command is then sent to the switches.

In addition to setting switches the new errors must be masked from the interrupt circuits; otherwise they will continue to generate interrupts. This is done in a manner similar to the way in which the switches were changed. A new mask image is created from the fault information and sent to the masked SCC's. Again, a clock signal is required to send the mask to the SCC's. This clock signal is enabled by the same code wordthat enabled the switch command clock.

The main error processing flow diagram is completed in Figure 23. Once reconfiguration is complete, or it is determined that no reconfiguration is needed, the processor is sent the address of the rollback routine. The roll-back routine causes the processor to resume execution of the applications program that was interrupted by the error. The execution starts at a prior point, where the state of the processors and all interim computations were saved. Thus, the computation is restarted at a point that was unaffected by the error; the execution is restarted by signals that start the processor clocks in synchronism.

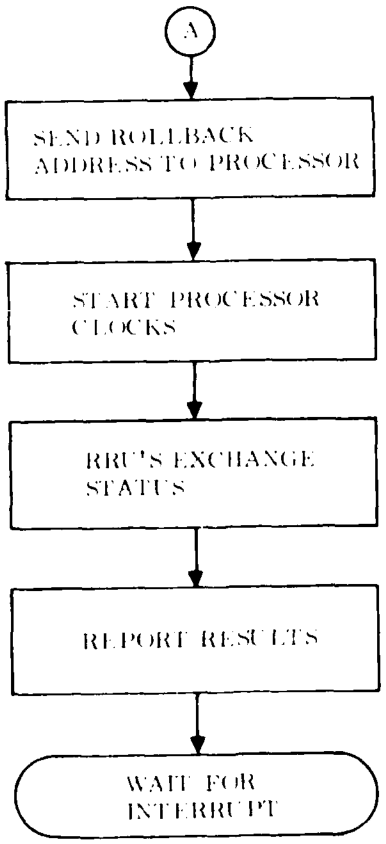


Figure 23. Final Part of RRU Operational Flow Diagram

The RRU's now exchange system status information off-line without interfering with the processor operation. The main purpose of this exchange is a form of self test to ensure that each processor reached the same result. Should there be any disagreement, the bad RRU processor will be declared down. This status exchange is also needed when failures occur in the interrupt system, since these errors are not normally monitored by the other RRU processors and a status exchange is the only way to update the status in the other RRU processors.

The results of reconfiguration are then reported to the maintainence system, which accepts status reports from each of the processors. If one differs it assumes the disagreeing RRU processor is bad. At this point the RRU's return to the idle or self-test loop they were executing when the original interrupt occurred.

3. Self Test Operations

The self tests described in this section are a form of built-in tests. They may be used at any time to determine the operating condition of either the processors or the RRU system. This is in contrast to reacting only to errors detected during application program execution, as described in the previous section. The self-test operation does require the cooperation of the bit-slice processors for all tests except the RRU processor diagnostic. Thus, the processors will not be available for execution of application programs during most of the testing. The self tests require many iterations of test sequences with resulting long execution times. The entire sequence of tests, if performed at one time, could last for several seconds.

The self-test capability would be used to determine the status of the system, before and after a mission, to ensure that there are no failed components. A second use for the self tests is to augment the error response operations. The error response operation cannot directly detect errors such as a partial RRU processor failure. It will diagnose such failures as a control failure in either a vote switch or as an SCC failure. The system will, however, be correctly reconfigured but it may take a longer time to determine the correct reconfiguration than it would have if the error had previously been detected by self test. An example best illustrates this point. Assume that the line in the I O port that controls a given voter switch module is broken and undiscovered. Then the voter fails as a second fault. The error response system will then attempt to switch to Channel 1 to bypass the voter. However, since the RRU is broken, nothing happens. As soon as the processors begins execution the voter causes another error interrupt. The error response system now assumes the channel - 1 switch is bad and tries to switch to channel 2. This sequence of restarts continues until all channels have been tried and the entire voter switch is declared down. If the self tests had been run, the bad switch control operation would have been previously detected so that, when the voter failed, the system could immediately declare the entire voter switch down and act accordingly.

The price for this added speed is a loss of processor time and additional program memory in the RRU's. The loss of processor time could be alleviated if there was a method of running the self tests periodically during the mission, when the processors were not needed for executing application software. This would, of course, depend on the exact nature of a given mission.

A flow diagram for the self tests is given in Figure 24. Each of the tests is summarized in Figures 25 through 27. The first test is the RRU processor self-test. This test does not require the main processor and so may be run concurrently with applications software. It is an instruction diagnostic that makes simple computations and compares the results to pre-computed values. Two bits on the I/O ports are provided for testing I/O instructions.

The next test is an exchange of data via the intercommunication I/O lines. It tests these lines as well as provide an opportunity for the RRU processors to update the status of the system in each RRU processor. This routine also runs concurrently with the main processor application software.

The voter/switch test, the RRU clock controller test, and the SCC error detection and masking test all require the cooperation of the main processors. These tests are similar in that they attempt to exercise all parts of the RRU system by using the main processors to provide known test vectors at all partition interfaces. Manipulation of these test vectors coupled with the reaction of the RRU system provides a means of diagnosing RRU failures. These tests all rely on the error response system to sense the state of the errors in order to make maximum use of that software. These tests are quite involved and lengthy and are split into segments so that only one segment need be executed at a time, thus minimizing the length of time that the main processor must be tied up performing self test.

## E. RRU DESIGN

### 1. Hardware Components

A block diagram of the RRU system is shown in Figure 28. The error reporting SCC's, while not shown in this diagram, are considered a part of the RRU. The functions and circuit details of the error reporting SCC's, the masked SCC's, the voter/switches and the main processors are described in detail elsewhere in this report. The only description of these modules in this section will be to describe their interface with the RRU processors.

The central control of the RRU lies in the RRU processors. Their primary function is to read the state of the system as inputs through their input ports and then compute the appropriate control signals to be output via their output ports.

In addition to the RRU processors and its I/O ports there are triplicated RRU clock controllers and main processor controllers. The RRU clock controller's function is to limit access to the control of the error reporting SCC registers, the masked SCC mask registers and the voter/switch switching commands. It does this by gating all the clocks entering these functions from the RRU processors. The reason for providing this function is to reduce the probability that a failed RRU processor could inadvertently fail the entire system by randomly manipulating the SCC's and voter/switches. There are three ways of enabling these clocks, all of which require a different code word to be calculated by the RRU. The first method uses the voted outputs of the interrupts from all three channels. Thus, anytime a normal error interrupt occurs the clocks can be enabled with a code word. The second method is to enable

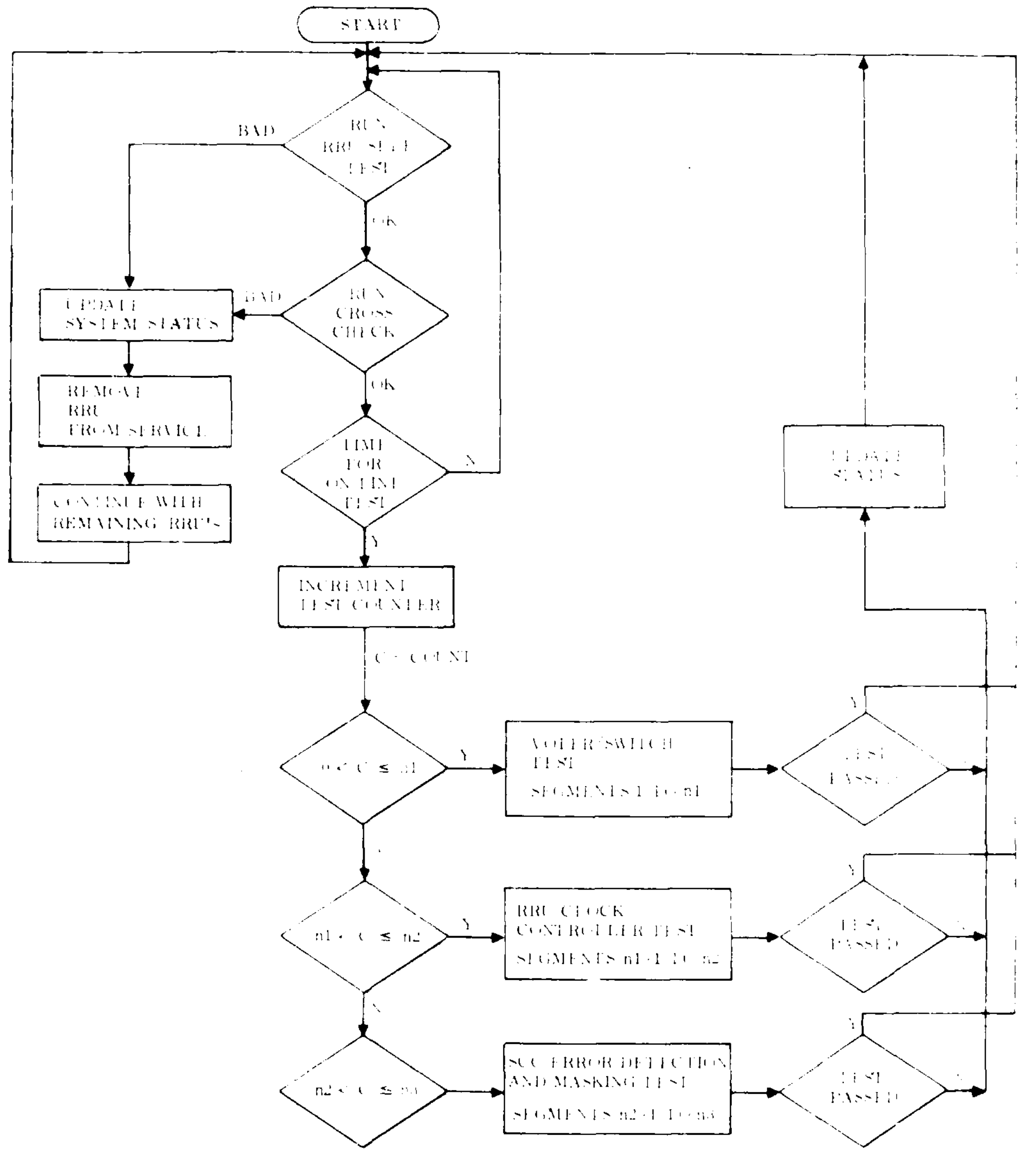


Figure 24. Self-Test Diagram

| TEST NAME | DESCRIPTION | COMPONENTS TESTED |
|---|---|---|
| RRU PROCESSOR SELF TEST | - INSTRUCTION DIAGNOSTICS<br>- CONTROL STORE COPY CHECKSUM | - RRU PROCESSOR<br>- I/O TEST LINES |
| RRU CROSS CHECK TEST | - RRU PROCESSORS COMPARE CODE WORDS<br>- USED TO UPDATE ANY STATUS CHANGE DUE TO SELF TEST RESULTS | - RRU PROCESSOR INTERCOMMUNICATION INPUT & OUTPUT LINES |

Figure 25. RRU Off Line Self Tests

| TEST | DESCRIPTION | COMPONENTS TESTED |
|---|---|---|
| VOTER/SWITCH | - TURN ON RRU CONTROL CLOCKS USING TEST LINES<br>- SINGLE STEP MAIN PROCESSORS EXECUTING A EST PROGRAM THAT PUTS KNOWN DATA AT ACH POSITION INTERFACE.<br>- MAKE DATA ON 2 CHANNELS EQUAL, 3RD DIFFERENT.<br>- CHECK VOTERS AGAINST OTHER SWITCHES FOR NO ERRORS WITH ALL COMBINATIONS OF 1 CHANNEL DIFFERENT.<br>- MAKE ALL CHANNELS DIFFERENT<br>- CHECK VOTERS AGAINST SWITCH OUTPUTS IN OTHER CHANNELS. SHOULD CAUSE ERRORS.<br>- MAKE DATA ON 2 CHANNELS EQUAL. 3RD DIFFERENT.<br>- TEST ALL SWITCH POSITIONS BY OBSERVING APPROPRIATE CHANNEL ERRORS<br>- REPEAT TEST WITH INVERTED DATA TO FIND STUCK AT FAULTS. | - RRU TEST LINES<br>- V/S CONTROL LINES<br>- V/S CLOCK LINES<br>- SCC LINES<br>- CLOCK CONTROLLER LINES<br>- V/S CIRCUITS<br>- SCC REGISTERS<br>- SCC OUTPUTS<br>- SCC CLOCKS<br>- RRU CLOCK CONTROLLER |
| RRU CLOCK CONTROLLER TEST | - RUN VOTER/SWITCH TEST ABOVE<br>- USE VOTED INTERRUPTS TO ENABLE RRU CLOCK CONTROLLER<br>- USE SINGLE INTERRUPT TO ENABLE RRU CLOCK CONTROLLER | - NORMAL RRU CLOCK ENABLE<br>- SINGLE CHANNEL CLOCK ENABLE |

Figure 26. RRU On Line Self Tests (Voter Switch & Clock Controller)

| TEST | DESCRIPTION | COMPONENTS TESTED |
|---|---|---|
| SCC ERROR DETECTION AND MASKING | - TURN ON RRU CONTROL CLOCKS USING TEST LINES<br><br>- SINGLE STEP PROCESSORS EXECUTING A TEST PROGRAM THAT PUTS KNOWN DATA AT EACH PARTITION INTERFACE<br><br>- PUT 1 BIT ERRORS IN EACH OF THE 16-BIT POSITIONS.<br>- TEST FOR APPROPRIATE SCC ERROR SIGNALS<br>- ALSO TEST PRESENCE OF INTERRUPT SIGNALS<br>- RUN 1 TEST WITH NO ERRORS AND OBSERVE NO INTERRUPT SIGNALS<br>- RERUN TESTS WITH ALL MASKS SET. NO INTERRUPTS SHOULD OCCUR | - SCC ERROR OUTPUTS<br>- RRU INTERRUPT SIGNALS<br>- SCC MASK CIRCUITS<br>- RRU PROCESSOR<br>● LINES TO MASKS<br>● LINES FROM SCC<br>● INTERRUPT LINES |

Figure 27. RRU On Line Self Tests (SCC)

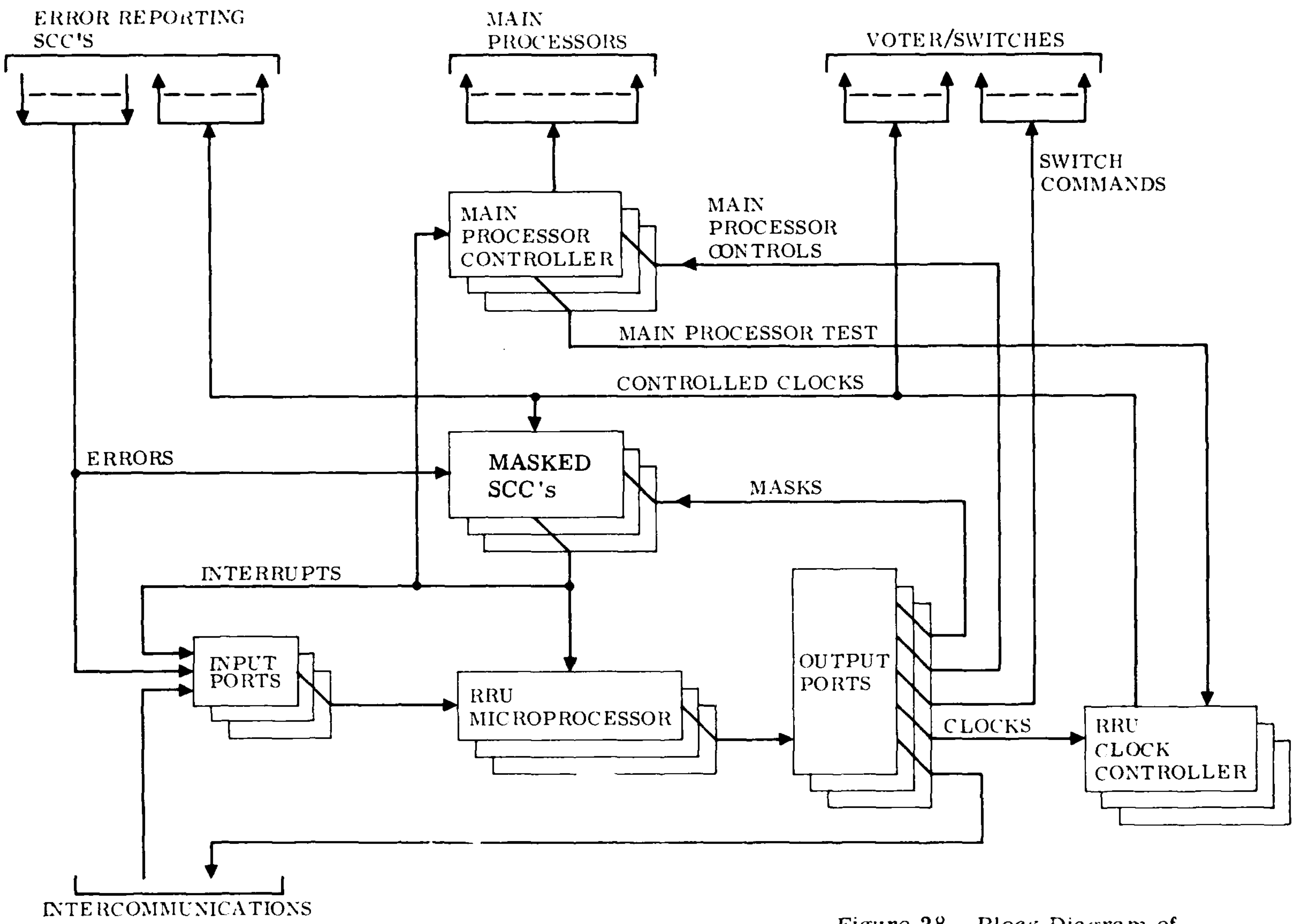


Figure 28. Block Diagram of RRU System

them when running tests by voting test control signals from all three RRU processors. This, together with the RRU processor-generated code word and a test signal from the main processor, will allow generation of clocks for testing. The last method enables the clocks when two of the RRU processors are down. It does this with just the good processor's interrupt signal and a code word. The RRU clock controller is described in detail in Section III.G.

The main bit-slice processor controller is used to start and stop the main processors. Its function includes turning off the bit-slice processor clocks on the receipt of an error interrupt from the SCC's. The controls necessary to force the processors to start at any one of a number of test routines or to go to the rollback point are also provided. These controllers consist of a few gates and flip-flops.

The RRU processors are implemented with monolithic microprocessors. The baseline system used Intel 8048 single-chip microprocessors, which have their own clock oscillators and require only the addition of a crystal and two capacitors. It also contains 1000 bytes of ROM, 64 bytes of RAM and a timer on the same chip. The 8748 is another version of this microprocessor, which has 1000 bytes of EPROM rather than ROM. The 8048 microprocessor is intended for control purposes and its I/O port capability is easily expanded to large numbers of I/O pins. It has a cycle time of 2.5 microseconds. Most instructions require only one cycle but some do require two cycles. It is a 40-pin device.

The I/O ports used in the baseline are Intel 8355 combination ROM and I/O ports. They contain 2000 bytes of ROM and two 8 bit I/O ports, which can be configured on a bit basis to be either input or output pins. Data placed on the outputs can also be read so they form a convenient means for determining which data is being outputted. There is an EPROM version designated the 8755.

The 8355 ROM and I/O circuits are connected in a straightforward fashion to the 8048. All of the necessary signals to drive the 8355's are present on the 8048. The 8355's require very little drive current and, therefore, line driving buffers are not needed. An example of an 8048 connected to one 8355 is shown in Figure 29.

Alternatives to the 8048 microprocessor are either the Intel 8044 or 8085. The 8049 contains twice the amount of ROM and RAM but is otherwise identical to the 8048. The 8085 is a version of the 8080 microprocessor that has a self-contained oscillator and timer and is compatible with the 8355 memory and I/O device. Both the 8049 and 8085 are available in faster versions than the 8048. The instruction repertoire for the 8085 is different from the 8048 so it is difficult to determine how much faster it might be in this application. Both of these microprocessors are candidates for use if greater speed is desired.

## 2. Description of RRU Signals

The details of the RRU design are probably best understood by describing the function of each signal going to and from the RRU processor. The input signals are listed in Figure 30 and the output signals are listed in Figure31. Signal positions in the list will be designated by an I for input signals or an O for output signals, followed by a letter designating the major heading and then by its number under the heading. Therefore, the number 1 output SSC signal is designated IA1.

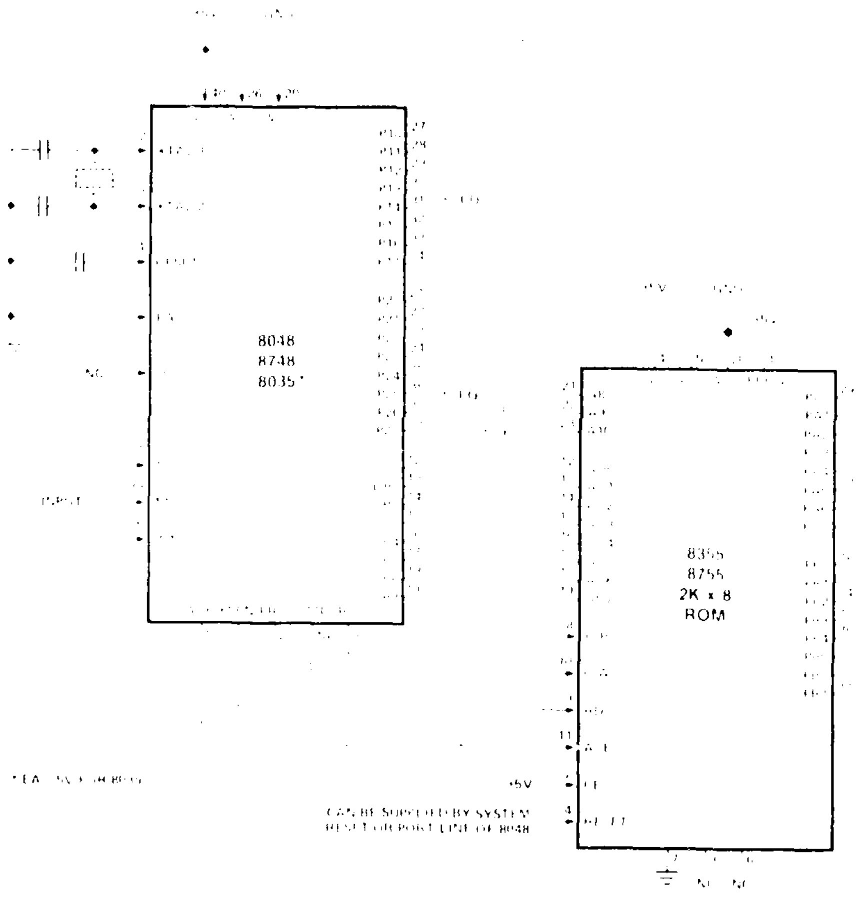


Figure 29. Microprocessor and Combination ROM and I/O Port Interconnections for Baseline Components

A. SCC ERROR/REGISTER OUTPUTS

1. 1 - OUT
2. 1 - IN
3. 2 - OUT
4. 2 - IN
5. 3a - OUT
6. 3a - IN
7. 3b - OUT
8. 3b - IN
9. 3c - OUT
10. 3c - IN
11. 4a - OUT
12. 4b - IN

B. INTERCOMMUNICATIONS

1. READY A IN
2. READY B IN
3. READY P IN
4. SEND/RECEIVE A IN
5. SEND/RECEIVE B IN
6. SEND/RECEIVE P IN
7. DATA A IN
8. DATA B IN

C. INTERRUPT

1. INT 1
2. INT 2
3. INT 3

D. I/O INSTRUCTION TEST

1. I/O TEST IN

E. PROCESSOR CLOCK CONTROLLER

1. TEST A
2. TEST B

Figure 30. Input Signals

A. S.C.C. MASKS

1., 2. MASK 1
3., 4. MASK 2
5., 6. MASK 3

B. VOTER SWITCH CONTROL

1. V/S 1
2. V/S 2
3. V/S 3a
4. V/S 3b
5. V/S 3c
6. V/S 3d
7. V/S 3e
8. V/S 3f
9. V/S 4a
10. V/S 4b

C. INTERCOMMUNICATIONS

1. READY A OUT
2. READY B OUT
3. READY P OUT
4. SEND RECEIVE OUT
5. DATA OUT

D. MISC. CONTROL

1. SCC REGISTER SELECT
2. CLEAR
3. I/O INST. TEST

E. RRU CLOCK CONTROLLER

1.–4. CODE WORD
5. CODE WORD CLOCK
6. MASK CLOCK
7. V/S 1-3f CLOCK
8. V/S 4a+4b CLOCK
9. SCC 1-3b CLOCK
10. SCC 3c-4b CLOCK
11. RRU TEST CONTROL

F. MAIN PROCESSOR CONTROL

1.–5. INTERRUPT VECTOR
6. RRU INTERRUPT BLOCK
7. CONTINUE
8. A CHANNEL BYPASS
9. B CHANNEL BYPASS

Figure 31. Output Signals

The first set of signals to be discussed are the RRU clock controller signals. As mentioned previously the function of the clock controller is to limit inadvertent access to those modules controlled by the RRU processor. The signals OE1 through 4 are the lines over which the RRU processor sends a code word to the controller. Four parallel lines are used to speed up the sending of the code word. The code word requires a clock to enter it into the controller. This clock is signal OE5 and is simply another output pin, which is alternately set to one and zero by software in the RRU processor.

The next five signals are the actual clocks that are used to clock masks to the masked SCC's, change the settings of the voter switches and read the error SCC registers. The last signal OE11 is a signal that is voted with corresponding signals from the other RRU processors to enable the clocks for test purposes.

The interrupt signals IC1-3 are the three interrupt signals generated by the masked SCC's. These signals are provided for testing purposes. The interrupt system is shown in Figure 32. Each set of masked SCC's generates an interrupt signal. All three signals are supplied to three switches each of which supplies an interrupt to one of the RRU processors. Should this signal fail such that it creates an interrupt, the interrupted processor will perform the normal error routines and find no errors to account for this interrupt. It then samples the INT signal that was passing through its interrupt switch. If that signal is high then the problem is in the Masked SCC's and the switch can be changed to use one of the other interrupt signals. If the INT signal was low then the error is between the masked SCC's and the RRU processor,and nothing more can be done to reconfigure the system; then that RRU will be declared down. The other two INT signals are provided for those cases when they are the signals used to interrupt the RRU processor. The opposite problem -- no interrupt is generated when it should be -- is detected when status is exchanged between processors. At this point the affected processor can test the INT lines and reconfigure as previously described.

Figure 32 also shows signals going from the RRU processors to the masked SCC's. These are the mask signals OA1-6. A mask image is sent serially in two-rail fashion over two lines. The data are entered by means of a mask clock, OE6, from the clock controller.

There is one line from each of the error reporting SCC's going to each RRU processor. They are input signals IA1-12. The errors as well as the register contents are read from these lines by clocking them with the appropriate clock signal, OE9 or 10, from the RRU clock controller. There is an additional control, OD1, called SCC register select. This connects either the i-1 register or the i registers to the error SCC output. This is important when a control store error has occurred. The SCC register at the control store output must output the current or i value since it is this value that must be checked for errors. However, the SCC register at the input to the control store must also be read to obtain the address to be used to look up the correct output in the control store copy in the RRU processor. The address that caused the error was the previous or i-1 value, which is also stored in the registers. If the SCC output was not switched to the i-1 register by the register select bit the RRU processor would have to read through the 48 bits of the i value first, and this would slow the processing considerably.

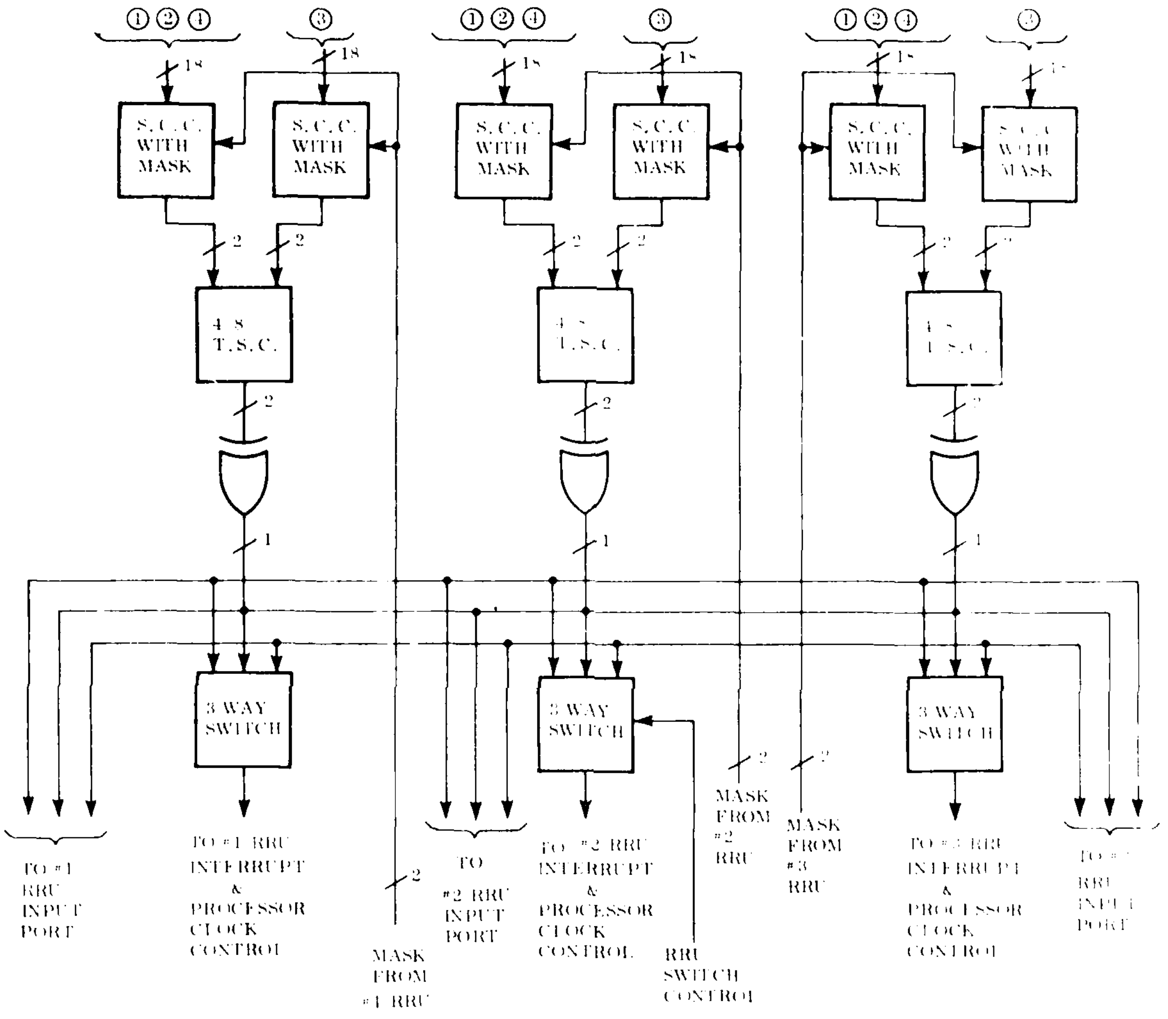


Figure 32. RRU Maskable S.C.C. Tree

The voter/switch commands, to enable either the voter or individual channel inputs, are sent to the voter/switches via signals OB1-10. The command is sent serially out to the appropriate line by clocking the voter/switches with either the OE7 or 8 clock signals, which pass through the RRU clock controller.

There are several input and output lines used to communicate between the RRU processors and the maintenance panel. The input signals are IB1-8 and the output signals are OC1-5. The ready-in, ready-out and send/receive in and out signals are used as handshake signals between RRU processors and the test panel. The ready-in signal signifies that the other processor is ready to transmit or receive, depending on the state of the send/receive line. The ready out and send/receive out are the handshake signals being sent out by the processor. Only one send/receive out signal is needed since it can be bussed to all the other processors. There must be separate ready-out signals so that only one processor responds at a time. The data-in lines come only from the two other processors, since the maintenance panel does not transmit data. The data-out signal is bussed to all the other processors. The processors normally stay in the receive mode until they wish to transmit. Should the processor that was to receive a message switch to the transmit state at the same time as the first processor, the handshakes will prevent the transmission from occurring. Normally the order of transmissions are fixed in the programs so that this problem does not arise.

The main processor clock control uses signals OF1-9 and IE1-2. The signals OF1-5 comprise an interrupt vector, which is sent to the main processor to cause it to jump to its next routine. It is used to send the main processor either to a rollback point or to a test routine. The processor is then restarted using the continue signal. If a test program is to be run, the RRU interrupt block signal, OF6, is enabled. This signal prevents the main processor from being stopped, before the end of a diagnostic, by errors early in the diagnostic. It is important that this be done because the RRU can only interpret test values at the end of a diagnostic. The bypass and test signals determine whether the other RRU processors are failing to issue a continue signal, in which case it can be bypassed.

The I/O test signals, ID1 and OD3, are used by the RRU processor self-test diagnostic to test its I/O instructions, by sending data out one pin and reading via the other pin. It can accomplish this without disturbing the data currently on the other output lines.

The last signal, OD2, is a clear signal which can be used by the RRU processor to initialize the various circuits in the system.

## 3. Maskable SCC Tree

The function of the Maskable SCC Tree of each of the RRU error-processing channels is to reduce the large number of signals (12) to a single signal as shown in Figure 32. The SCC outputs of interfaces ① ② and ④ (see Figure 13 for numbered interface) can be accommodated by one SCC with Mask, while the SCC outputs of the interface (3) are delivered to the second SCC with Mask, since each of these SCC's can handle 24 inputs or less. The outputs of each of these devices is combined in a second checker, a four-out-of-eight checker whose dual-rail output is converted to single-rail form by an exclusive OR circuit. This is done to match the RRU's interrupt signal requirements.

These SCC with Mask devices differ significantly from those used in the bit-slice processor interfaces in two ways. One of these is the wider input capability, as shown in Figure 33. Another is the incorporation of a masking capability to block signals known to produce error signals from descending the tree and producing undesired error indications. The mask is computed by each tree's RRU computer, based on the current error state of the SDFTP, and is usually changed after the detection of a solid error. By having the RRU update the mask after each detection, it is possible for the SDFTP to continue operating after a number of errors have been detected, without the known faults continually interrupting the SDFTP every time a vector occurs that exercises one of these faults.

The outputs of all three of the SCC trees E-OR gates can be selected to interrupt any of the RRU computers by means of a multiplexer as shown in Figure 32. These three E-OR outputs are made available to the RRU computer by connecting them to the RRU computer's input buffers. This permits each RRU computer to isolate failures to the SCC trees.

## F. RRU PROCESSOR SOFTWARE

### 1. Design

The RRU processor software is divided into two parts. The first part implements the error response processing. This processing was discussed in Section III.D and its operational flow diagrams are contained in Figures 19, 21, 22, and 23. The second part implements the self-test processing discussed in Section III.D and is diagrammed in Figure 24. The function of the error response sys. [illegible] is to react to errors reported in the self-checking checkers, reconfigure the main processors, if necessary, by controlling the switches, and report the fault to the maintenance system. The self-test system provides a means of testing portions of the RRU that cannot be guaranteed to be directly tested by the self-checking checkers. This function is useful for built-in test functions, to ascertain the health of the system before a mission, or to supplement the error response system by providing information that may in some instances speed up the error response processing.

Whenever possible, common routines are made into subroutines to save program memory space. This usually results in slightly longer running times so that only the longer routines are made into subroutines.

The sizing estimates and running times for those parts of the system that require software diagnostics cannot be accurately estimated, for it is beyond the scope of this work and is too early in the design to derive accurate estimates of coverage (percent of circuitry tested) versus program size and running time. The attendant problems are discussed in some detail in Section IV.

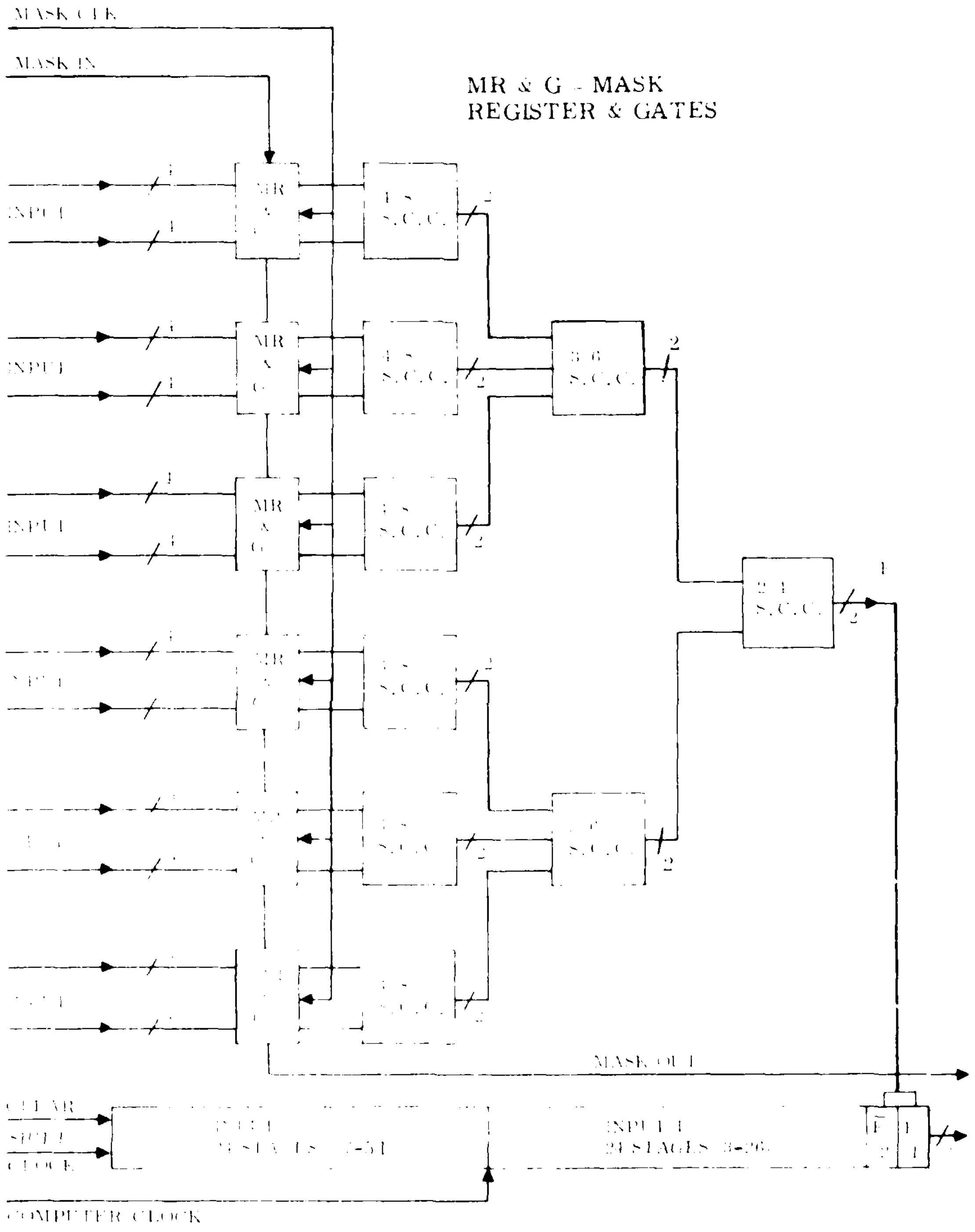


Figure 33. S.C.C. With Mask

### 2. Program Size Estimates

Estimates of the program sizes for both the error response and self-test software are given in Figure 34. The error response software requires 3200 bytes of memory, the self-test software slightly less, at 3000 bytes. However, it is not accurately known how much coverage is obtained for this amount of code. The control store tables in item 3 provide a fourth copy of the main processor's control store to help determine which control store is still functioning after two of the three have failed.

The total program storage is 9200 bytes. Each RRU processor has 11,000 bytes of ROM. However, 4000 bytes are connected as data memory to access the control store tables more efficiently. The control store tables require only 3000 bytes, leaving 1000 bytes unused. If this memory were to be used as program storage, a few more gates would be needed to give access to it as program memory. Currently, without this 1000 bytes, there are still 800 bytes of spare program memory.

The self-test software uses the error response software to a large extent in performing tests. Thus the 3000 bytes for the self-test software is, in reality, the amount of additional software needed to perform the self-test function.

| | | ESTIMATED BYTES PER PROCESSOR |
|---|---|---|
| 1. | ERROR RESPONSE | 3200 |
| 2. | SELF TEST | 3000 |
| 3. | CONTROL STORE TABLES | 3000 |
| | TOTAL | 9200 |

Figure 34. RRU Program Size Estimates

### 3. Error Response Timing Estimates

The running time of the error response routines is largely dependent on the exact combination of failures in the system at any one time, because of the many different branches in the program logic. Therefore, the running time estimates are illustrated by five examples given in Figure 35.

The first two examples are the same as those illustrated in the error maps in Figure 17. The first example is the failure of the number 1 microsequencer with no other failures in the system. The software requires each failure

## 2. Program Size Estimates

Estimates of the program sizes for both the error response and self-test software are given in Figure 34. The error response software requires 3200 bytes of memory, the self-test software slightly less, at 3000 bytes. However, it is not accurately known how much coverage is obtained for this amount of code. The control store tables in item 3 provide a fourth copy of the main processor's control store to help determine which control store is still functioning after two of the three have failed.

The total program storage is 9200 bytes. Each RRU processor has 11,000 bytes of ROM. However, 4000 bytes are connected as data memory to access the control store tables more efficiently. The control store tables require only 3000 bytes, leaving 1000 bytes unused. If this memory were to be used as program storage, a few more gates would be needed to give access to it as program memory. Currently, without this 1000 bytes, there are still 800 bytes of spare program memory.

The self-test software uses the error response software to a large extent in performing tests. Thus the 3000 bytes for the self-test software is, in reality, the amount of additional software needed to perform the self-test function.

| | | ESTIMATED BYTES PER PROCESSOR |
|---|---|---|
| 1. | ERROR RESPONSE | 3200 |
| 2. | SELF TEST | 3000 |
| 3. | CONTROL STORE TABLES | 3000 |
| | TOTAL | 9200 |

Figure 34. RRU Program Size Estimates

## 3. Error Response Timing Estimates

The running time of the error response routines is largely dependent on the exact combination of failures in the system at any one time, because of the many different branches in the program logic. Therefore, the running time estimates are illustrated by five examples given in Figure 35.

The first two examples are the same as those illustrated in the error maps in Figure 17. The first example is the failure of the number 1 microsequencer with no other failures in the system. The software requires each failure

|  | Estimated Time in Microseconds |  |  |  |
|---|---|---|---|---|
|  | 1st Error | 2nd Error | Sum of 1st & 2nd | Off Line Status Reporting |
| 1. Microsequencer 1 Failure | 1800 | 2000 | 3800 | 16,400 |
| 2. Microsequencer 1 Prior Fault Plus a Current Failure in V/S 2 SCC 4 (Port 2) | 2400 | 3400 | 5800 | 16,400 |
| 3. Microsequencer 1 Prior Fault Plus a Current Failure in V/S 2 SCC 2 (Port 1) | 2100 | 3100 | 5200 | 16,400 |
| 4. Microsequencer 1 Prior Fault Plus a Current Failure in Microsequencer 2. This is a second fault in a partition. | 2100 | 8200 | 10,300 | 16,400 |
| 5. Microsequencer 1 Prior Fault Plus an Interrupt Circuit Failure | 1400 | 2200 | 3600 | 16,400 |

Figure 35. Error Response Timing Estimates

to occur twice before it is detected as a fault. This prevents transient errors from needlessly reconfiguring the system. Thus a failure of the number 1 sequencer will cause the error response software to take 1.8 milliseconds to process the first occurrence of the error. If the error is a hard failure, the system will immediately be interrupted with the second error, which takes two milliseconds to process. Thus, the main processors will have been off-line for the sum of these two times as indicated in the third column in Figure 35. This example requires a total of 3.8 milliseconds before the main processors are restarted at the previous rollback point.

If this error were only a transient then only the 1.8 milliseconds for the first error would be taken away from the main processor.

The RRU then takes an additional 16.4 milliseconds to report the failure and checks that all RRU processors have reported the same failure. This reporting process is the same for all tests.

The second example adds a Voter-Switch error to the number 1 microsequencer failure. The errors are reported through a different port for this Voter-Switch, slightly affecting the timing. The time to process the two errors is 5.8 milliseconds.

The third example is similar to the second example, except that the Voter-Switch error is reported through the same port as the microsequencer error, reducing the running time by 0.6 of a millisecond -- to 5.2 milliseconds.

The fourth example is an example of two failures in one partition, which is more difficult to handle. The processing time has almost doubled here because a microsequencer diagnostic must be run in the main processor. The total time required here is 10.3 milliseconds. The running time, in this case, is largely dependent on a main processor diagnostic running time, for which no accurate estimates are available.

The last example illustrates an interrupt circuit failure, where the interrupt is stuck in the "ON" state. This failure requires 3.6 milliseconds.

Thus the typical error-handling times range from 3.6 to 10.3 milliseconds.

## 4. Self Test Timing Estimates

The self test timing estimates are given in Figure 36. The RRU processor tests which take 23.7 milliseconds, are run without disturbing the main processor operation. The cooperation of the main processor is required to carry out the bulk of the testing on all the switches, self-checking checkers and other miscellaneous circuitry. The running time for these tests is given as item two in Figure 36. The complete set of tests is estimated to run for three seconds. This is not objectionable for built-in test requirements, but would be prohibited during a mission. These tests were made more useful by dividing them into small segments each of which could be run at different times. The typical length of these segments is given in line 3 as 3.6 milliseconds. This places each segment in the same time range as the error response routine times. Again, these estimates depend heavily on main processor diagnostics for which there are no accurate timing estimates available.

The last item is the 16.4 milliseconds needed for reporting each failure as it is detected.

| | | Estimated Time In Microseconds | |
|---|---|---|---|
| 1. | RRU Processor Tests (Main Processors on Line) | 23,700 | |
| 2. | Tests Involving Main Processor | 3,147,000 | (3 seconds) |
| 3. | Single Segment From #2 Above | 3,600 | |
| 4. | Status Reporting (Main Processors on Line) | 16,400 | |

Figure 36. Self Test Timing Estimates

## G. CUSTOM DEVICES

### 1. Summary

A number of circuits have been designed to facilitate the implementation of a self-diagnosing processor using dynamic redundancy. Most of the circuits are important in terms of the efficient realization of the concept and are predicated on large-scale integrated (LSI) circuit embodiment of a number of functions. The microsequencer device is desirable in any processor in that it reduces parts count and the number of devices in any processor. In the SDFTP, it is important because the microsequencer partition is replicated three times in the design and, thus, the parts reduction is increased by a factor of three.

The other four circuits are integral to the implementation of the self-diagnosis and fault tolerant concepts. Two self-checking circuits, which detect errors not only in their inputs but within the circuits themselves, are defined. They are the Self-Checking Checker (S.C.C.) Without Mask and With Mask. These LSI circuits are used between the bit-slice processor partitions and in the RRU respectively. The S.C.C. Without Mask is the realization of three totally self-checking checkers (T.S.C.) and three scanning registers. The S.C.C. With Mask accepts a larger input vector that can be masked, so that selected inputs are rendered ineffective in the code check of the device. However it has only a single TSC and therefore, unlike the S.C.C. Without Mask, cannot check the triplicated input.

The Voter-Switch is an important circuit with respect to the fault tolerance capabilities of the design because it provides the reconfiguration capability. It consists of a triplicated, voter and three-way switch, which can be used to connect triples of inputs to triples of outputs. In the voter, each output is the majority function of the three inputs. The choice of the interconnection can be externally selected.

The last circuit in the complement of these new LSI circuits, is the clock controller. It is applied in the RRU and is designed to prevent inadvertent clocking of information into the Voter-Switch devices and thereby, possibly changing the interconnection. The check signals are interlocked so that either the majority of the RRU computers control the clock and/or these computers supply a code word for one of three modes of operation.

2. Self-Checking Checker Without Mask

The Self-Checking Checker (S.C.C.) Without Mask is a device that checks the outputs of the SDFTP bit-slice partitions. It is designed to detect errors at its inputs as well as in the circuit itself. In addition to indicating the presence of an error at its outputs, it saves a copy of the input that is currently being checked as well as the previous input. The device is designed to check three sets of identical 16-bit inputs using self-checking circuits. The three sets of inputs are grouped into three different pairs of 16-bit inputs. Each pair of signals is encoded in a dual-rail unordered code, which is checked by a self-checking tree as shown in Figure 37. The two level tree is composed of four-out-of eight (4/8) self-checking checkers. The single second level (4/8) self-checking checker has a dual-rail output, encoded so that the 0·0 and 1·1 output combinations indicate that an error has been detected. The three trees are totally self-checking (TSC) in the sense that an assumed error never causes an erroneous output.

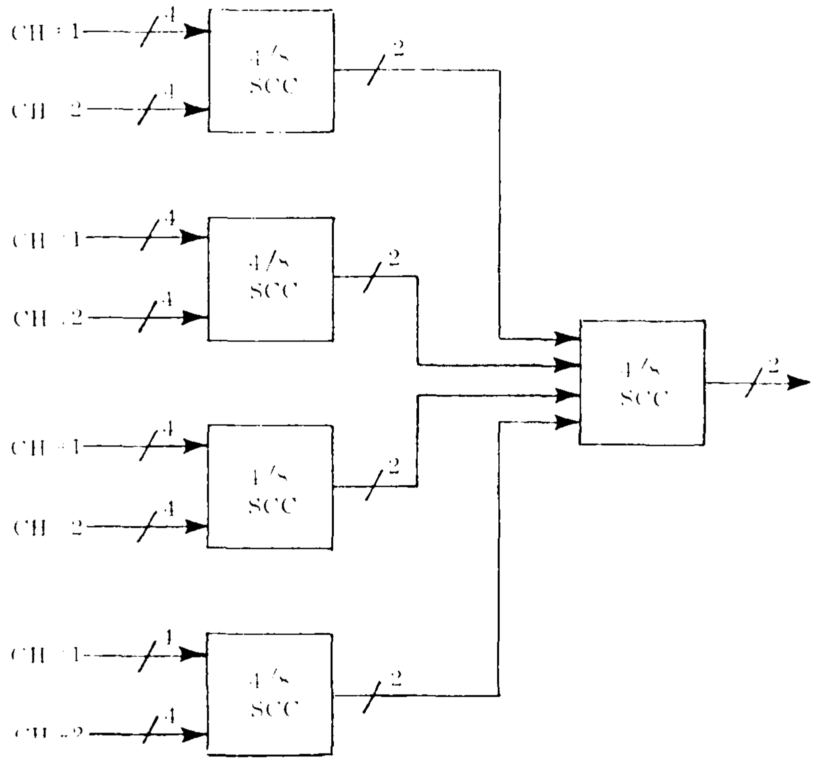


Figure 37. Totally Self-Checking Checker (TSC)

The outputs of the three TSC trees are saved in scanning registers. The three dual-rail outputs are clocked into the three scanning registers along with the three sets of 16 bit inputs as shown in Figure 38. Each of the register clock lines are brought out separately so that each of the scanning registers can be individually clocked as in the case of the SDFTP

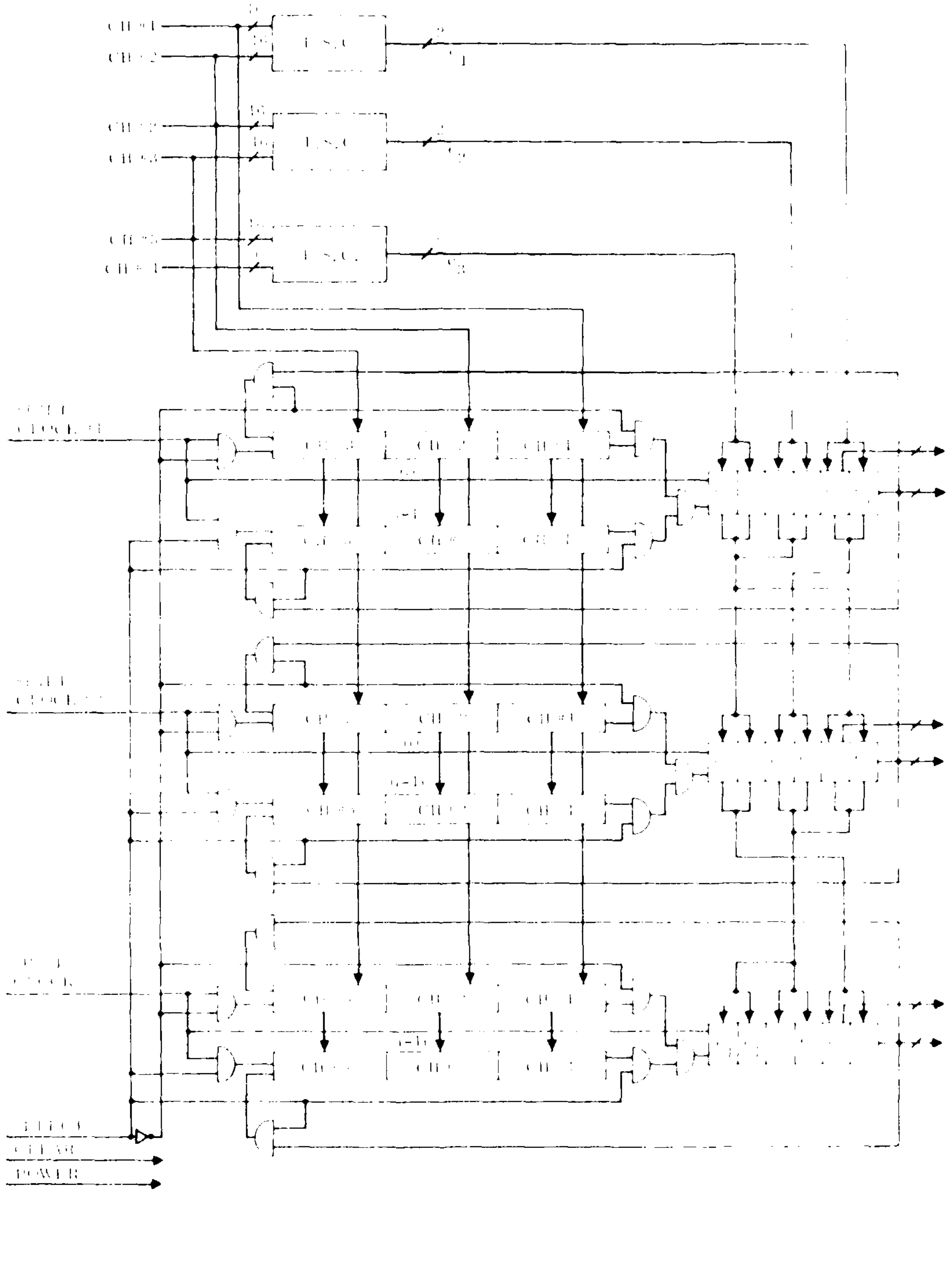


Figure 38. Self-Checking Checker (S.C.C.) Without Mask

The three scanning registers save the three sets of 16 bit inputs and the T.S.C. outputs; also the three sets of 16-bit inputs from the previous clock cycle are saved. This snapshot is updated every clock cycle with the current input sets shifted down to the previous cycle's locations. The format of the snapshot captured by the scanning register is shown in Figure 39. The three T.S.C. outputs are located at the output ends of the registers. The error information consisting of the outputs of the TSC, is rotated with respect to each other so that the three T.S.C. outputs are simultaneously available. This makes it possible to detect all errors detected by the T.S.C.'s as is done in the RRU S.S.C. With Mask.

The snapshots of the inputs and their T.S.C. outputs can be read out by supplying a clock signal to serially shift the data out. Since the shift clocks of the three scanning registers are separate, the shift out of the snapshots can be done independently by supplying the appropriate clock signals. The registers are designed to circulate the information as it is read out so that, if desired, the snapshot can be retransmitted by supplying additional shift pulses.

The S.C.C. Without Mask can accept up to three sets of 16 inputs and provides three dual-rail outputs. It is planned to be implemented in a 64-pin package and has a gate complexity of the order of 2200 gates. See Figure 40.

SCANNING REGISTER #1

| | | | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|
| INPUT 3 (i-1) | INPUT 2 (i-1) | INPUT 1 (i-1) | INPUT 3 (i) | INPUT 2 (i) | INPUT 1 (i) | $\bar{C}_3$ | $C_3$ | $\bar{C}_2$ | $C_2$ | $C_1$ | $C_1$ |

SCANNING REGISTER #2

| | | | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|
| INPUT 3 (i-1) | INPUT 2 (i-1) | INPUT 1 (i-1) | INPUT 3 (i) | INPUT 2 (i) | INPUT 1 (i) | $\bar{C}_2$ | $C_2$ | $\bar{C}_1$ | $C_1$ | $C_3$ | $C_3$ |

SCANNING REGISTER #3

| | | | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|
| INPUT 3 (i-1) | INPUT 2 (i-1) | INPUT 1 (i-1) | INPUT 3 (i) | INPUT 2 (i) | INPUT 1 (i) | $C_1$ | $C_1$ | $C_3$ | $C_3$ | $C_2$ | $C_2$ |

S.C.C. WITHOUT MASK

Figure 39. Scanning Register Format

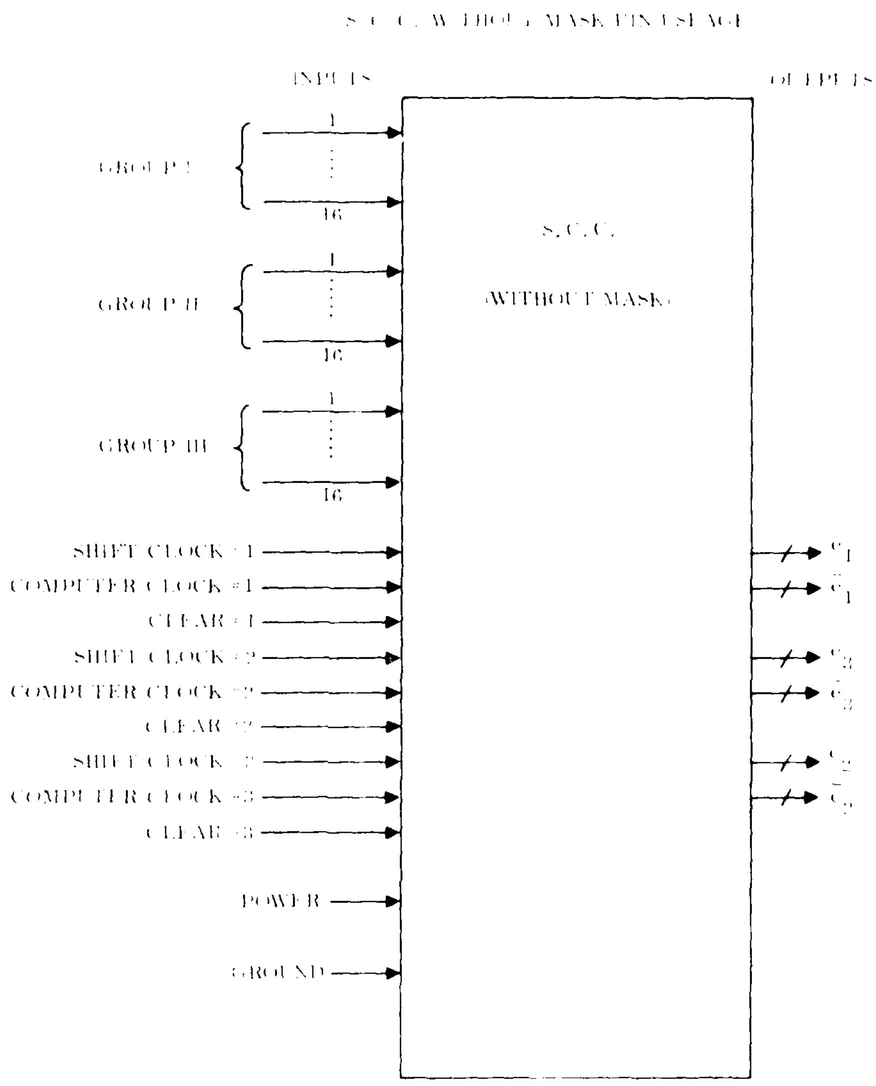


Figure 40. S.C.C. Without Mask Pin Usage

## 3. Self-Checking Checker (S.C.C.) With Mask

The Self-Checking Checker (S.C.C.) With Mask performs a function similar to the S.C. C. Without Mask; that is, it detects errors in its input and in the checker itself. Like the S.C.C. Without Mask the design is based on a Totally Self-Checking Checker (T.S.C.) tree. However, since the function of the S.C.C. With Mask, in the SDFTP, is to reduce the size of the SDFTP error vector, consisting of all of the S.C.C. Without Mask outputs, the input domain consists of pairs of signals where each bit is encoded in a dual rail code. Thus a single tree is sufficient to reduce this set of S.C.C. Without Mask outputs to a single dual-rail output having the same unordered code encoding. An error is indicated by the 0·0 or 1·1 signals.

This T.S.C. tree is a three-level tree. The first level is composed of 4 8 T.S.C.'s and this is followed by a pair of 3 6 T.S.C.'s. The last level, which produces the output, is a (2 4) T.S.C. As in the case of the S.C.C. Without Mask the output of the T.S.C. tree is saved in a scanning register as shown in Figure 41.

The T.S.C. is combined with the two dual-rail encoded 24-bit inputs to form the snapshot. This information is captured in the scanning register by using the computer clock line to set up the signals in the register flip-flops. In this device only the current inputs are saved as contrasted with the S.C.C. Without Mask device, which saves both the current input vectors and the immediately preceding clock-cycle inputs.

Since the function of this device is to reduce the number of S.C.C. Without Mask error signals, and it must continue to operate after one or more errors have been detected, those outputs that are known to be sources of errors must be masked out. Hence, a mask register, which can be externally loaded and will block the input signals that are not to participate in the over-all error generation (come from devices that have faults), is provided.

The blocked signals are replaced by properly encoded signals. A one in the mask blocks the input signal while a zero allows the input signal to be processed by the device circuits.

New masks are set into the mask register by serially shifting in the mask bits two at a time under the control of the mask clock. The operation of the mask part of the device can be verified by monitoring the Mask Out output when the shift clock is supplied. As new information is read in, the previous contents of the shift register are shifted out.

Pin usage is shown in Figure 42. Gate complexity is about 800.

4. Voter-Switch

The Voter-Switch device interconnects the SDFTP partitions. Since the partitions are triplicated the device must accept triplicated inputs and produce triplicated outputs. In the Voter-Switch device the triplication is at the bit level. Each of the triple outputs per bit is either the majority function of the triple inputs for the bit, or it is one of the triple inputs for the particular bit that has been selected by one of the Voter-Switch switches. Selection of the majority function or switch input is controlled by the Voter-OR-Switch (VOS) bit of the Voter-Switch Control Register. The selection of which of the three inputs is used is under the control of the switch control bits C1, C2 and C3, which are also stored in the Control Registers. To assure independence of each of the three sets of circuits,the Control Register is triplicated and is used to control only one of the three circuits that generate each bit's triplicated output. This is shown schematically in Figure 43 where a single bit of the Voter-Switch circuitry is shown with the three sets of Control Registers, which are used for all of the nine bits of input. The triplicated outputs for each bit are produced by three, three-level networks. The network output is developed by an OR-gate, which receives the outputs of the gated voter and the gated switch selected inputs. These inputs are gated by the second level of the network, which is controlled by the setting of the VOS bit of the Control Register.

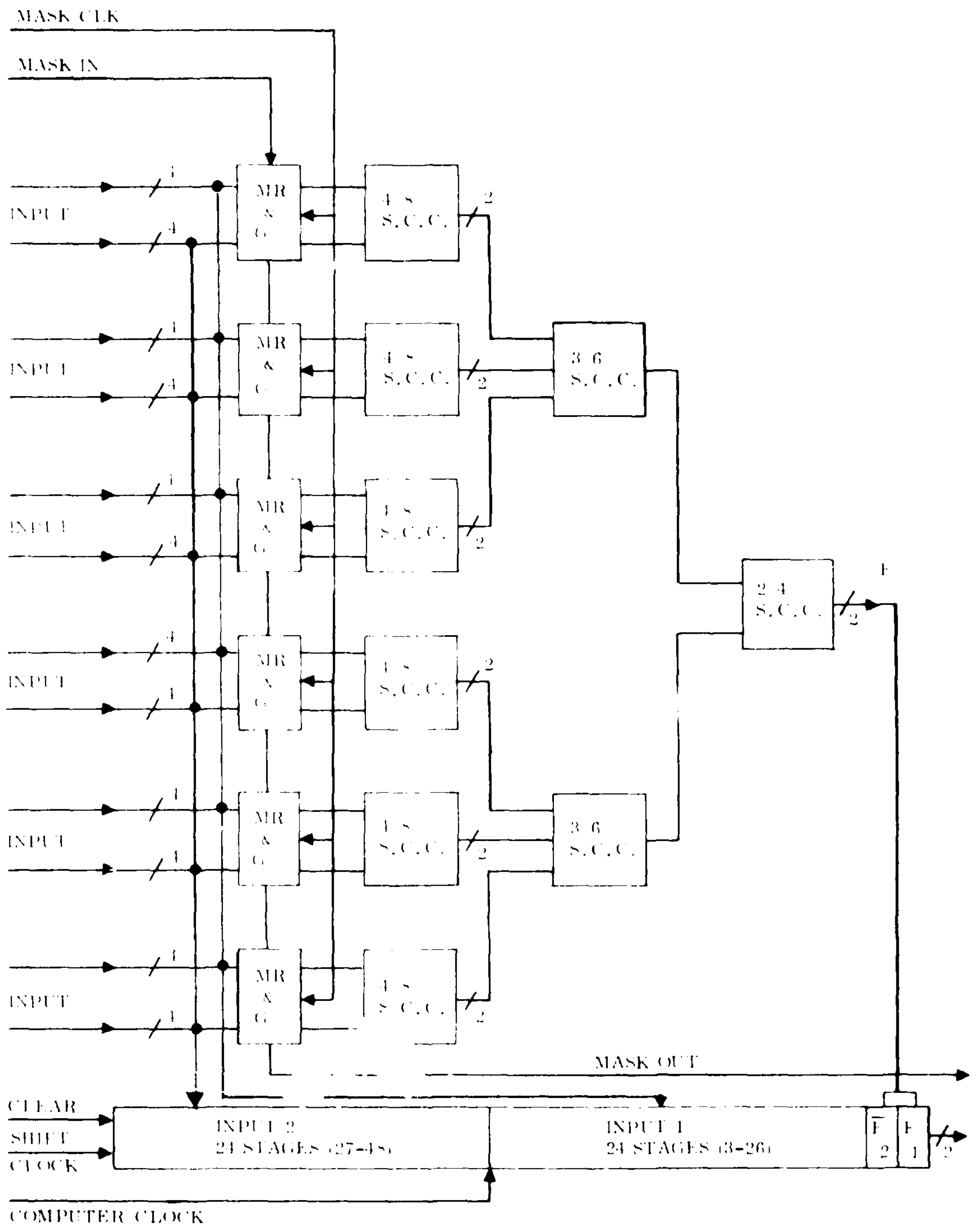


Figure 41. S.C.C. With Mask

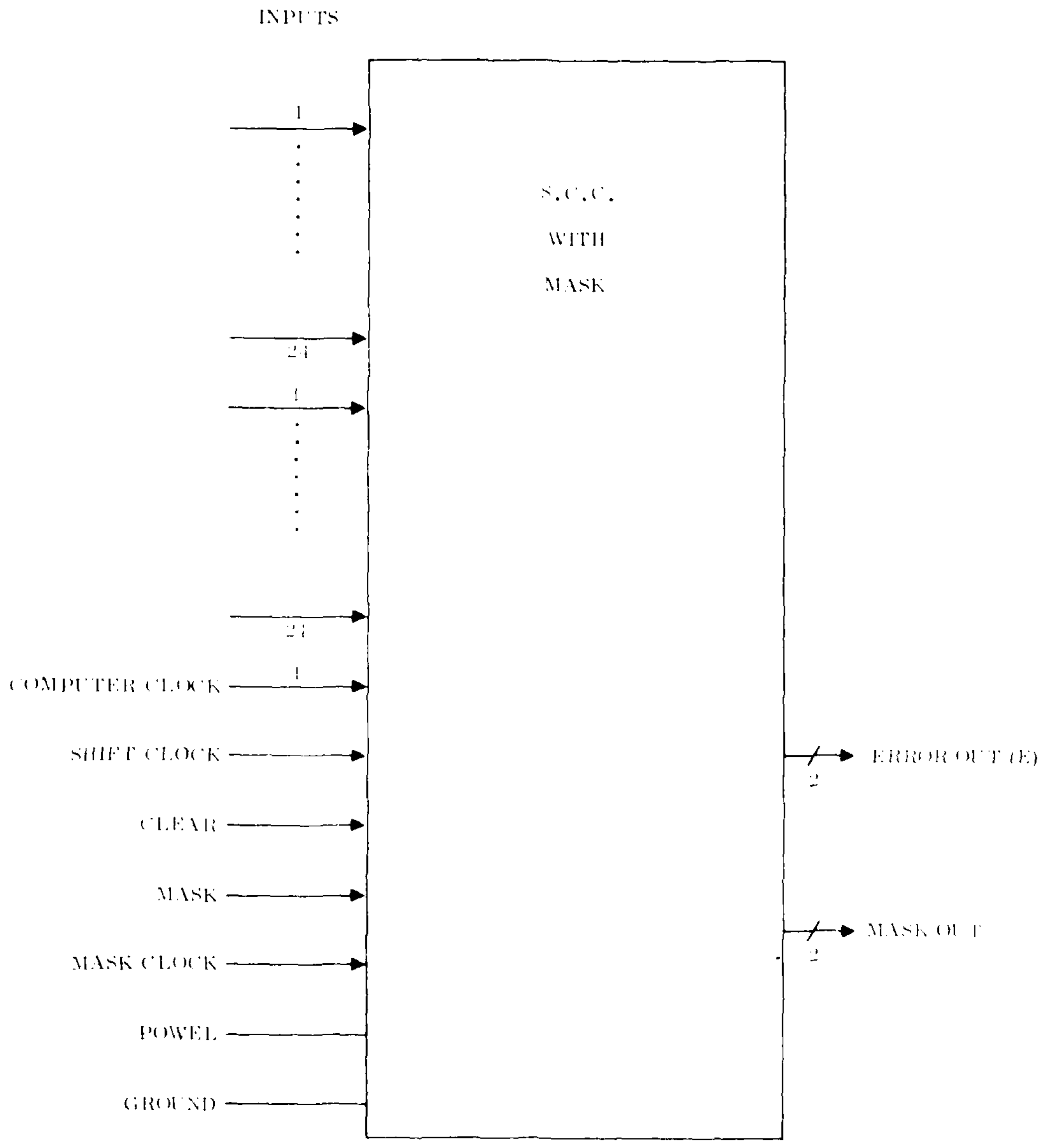


Figure 42. S.C.C. With Mask Pin Usage

A "one" value for the VOS selects the switches, while a "zero" selects the voter that produces the majority function of the inputs. The selection of the particular input, when the switches have been selected (the VOS is a one) is determined by the C bits of the Control Register. Setting C1 to a one selects i I 1 by means of the top, third level AND gate. Similarly, setting C2 and C3 to ones, selects i I 2 and i I 3, respectively. Hence, any input can be routed to any one of the three outputs for this bit and, therefore, can be accomplished in the other bit locations using their bit networks, which are identical. Excluding the Control Registers, the Voter-Switch consists of nine bits of triplicated networks as shown in Figure 43 for a total of 27 selector slices, which are identical except for their inputs.

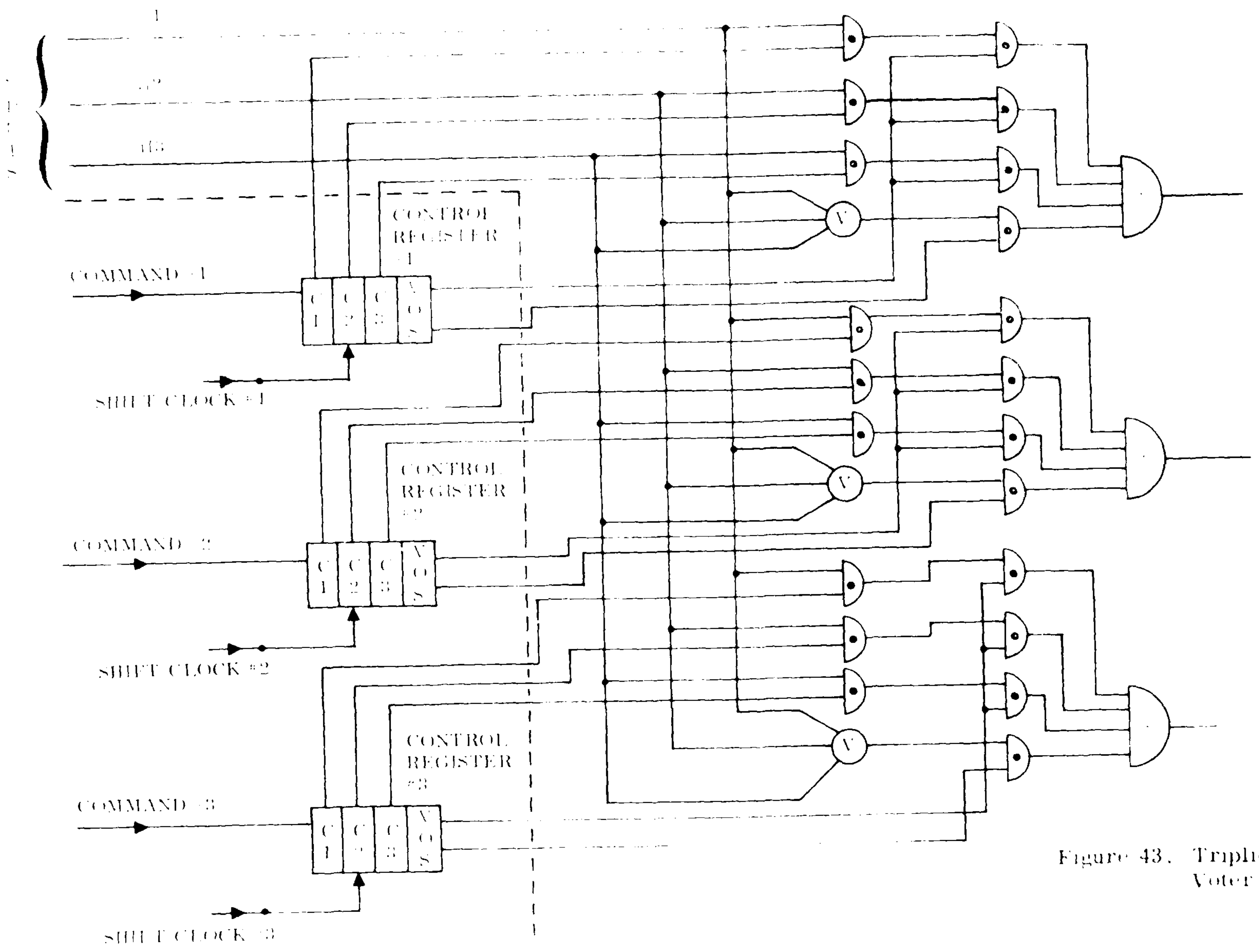


Figure 43. Triplicated Voter and Switch

Changes in the interconnection between partitions are accomplished by changing the three Control Register contents. Each Control Register controls one-third of the networks and each can be changed independently of the others. In order that these changes in the interconnections do not affect the information passing through the Voter-Switch, the changes should be scheduled during periods of time when information is not being exchanged between partitions.

Loading of the Control Registers is accomplished by serially shifting in the four bit commands using the shift-in clock. Since each Command Register has its own separate input and clock line the Command Registers can be individually loaded.

The Voter-Switch pin requirements are shown in Figure 44. The device complexity is of the order of 400 gates.

## 5. RRU Clock Controller

The RRU Clock Controller provides a comprehensive control over the RRU clock signals. It is designed to prevent inadvertent clock signals from changing the SDFTP control and diagnostic registers, except during certain prescribed intervals. The signals involved are the Self-Checking Checker Shift Clock, the Voter-Switch Command Clock, and the Self-Checking Checker With Mask Mask Clock. Besides regulating these signals the device provides additional drive for these signals.

Control is exercised over the clock signals by inhibiting each clock signal in a two-input AND-gate as shown in Figure 45. This inhibiting signal can be removed in three different ways, corresponding to three different modes of operation for the clock controller. All three modes are conditioned by a code word circuit that must receive the correct mode code word before the mode inhibit can be removed. On receipt of one of these code words and proper decoding, the designated mode can be commanded. This circuit is shown in the middle of Figure 45. It receives the code word a nibble (4 bits) at a time and shifts the 32-bit code word into the register under the control of the code word clock.

The first clock mode is the normal or unfailed RRU mode. In this mode, the three RRU S.C.C. With Mask signals are used to set flip-flops in which the outputs are voted by a majority gate as shown in the top of Figure 45. The flip-flops capture the S.C.C. With Mask error signal, since it will be lost as soon as the S.C.C. snapshot registers are read out. The voter output, which produces the majority function of the three RRU S.C.C. With Mask error signals, then changes the clock inhibit signal to an enable signal. The cascade of the two input AND and the OR provides the code word conditioning for this path (mode) and the alternative methods of enabling the output clock AND-gate.

The second mode is the single RRU interrupt signal which permits a single RRU computer to turn the clocks on after supplying the correct clock word. As in the first mode, the interrupt signal is used to set a flip-flop, which actually drives the clock enabling circuit consisting of a two input AND-gate and a three input OR (See Figure 45). Again, the AND gate provides the command word conditioning of this mode signal while the OR gate is used to merge the alternate clock enabling signals.

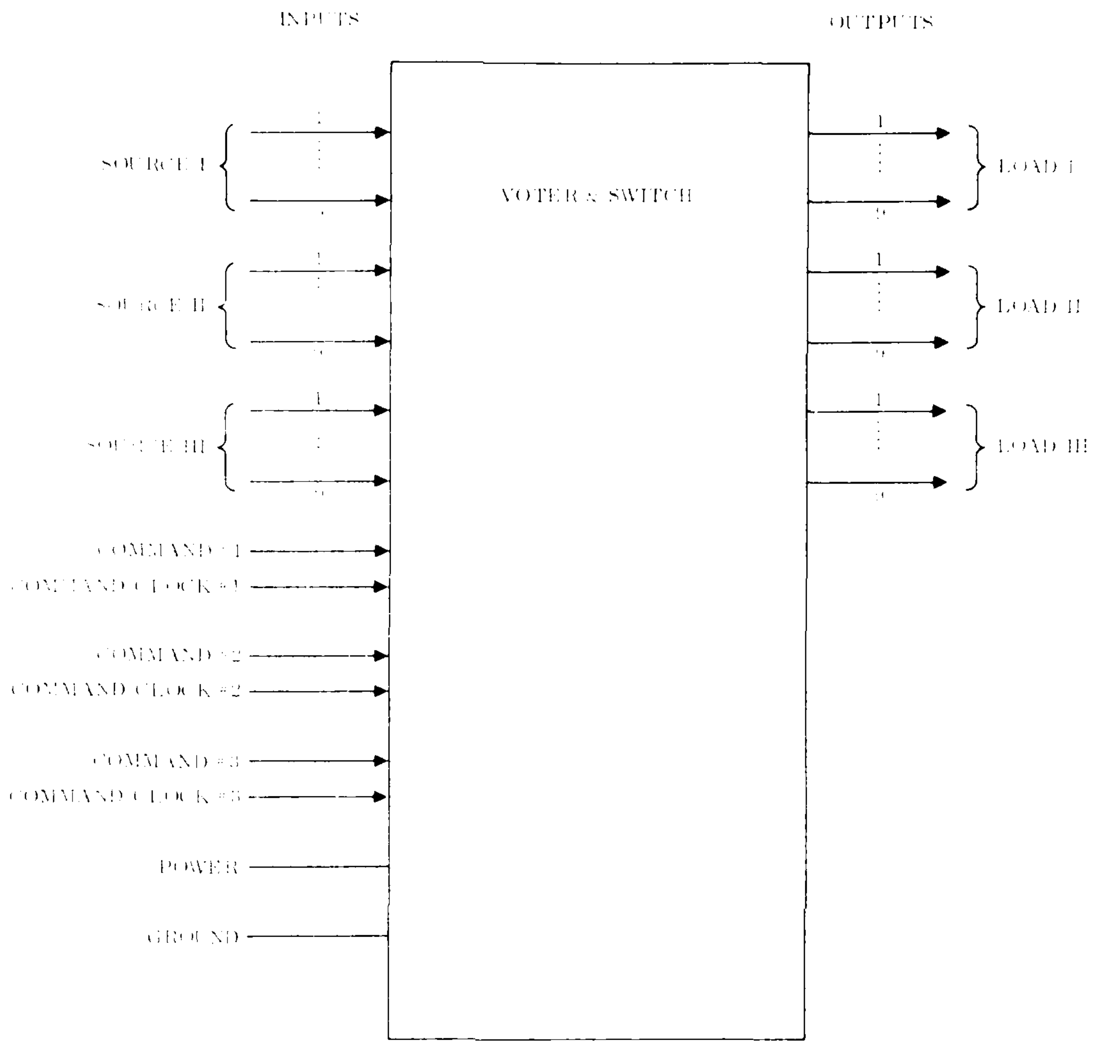


Figure 44. Voter and Switch Pin Requirements

The third test mode is provided to permit the RRU to be tested. The voter circuit at the bottom of Figure 45 produces the majority function of the three RRU computer test signals. The voter output is conditioned by both the code word decoder output and the main processor test signal so that testing can be performed only when both the RRU computer and the bit-slice processor, controlled by the RRU, have called for a clock turn-on. This conditioning is accomplished by the three-mode clock enable paths.

After each enabling of the clock signals, the circuit must be restored to its original inhibited condition by the reset signal. In the SDFTP this signal is generated by an RRU computer.

The clock controller device controls eight clock signals as shown in Figure 46. The proposed circuit is a 40-pin package and has complexity of about 400 gates.

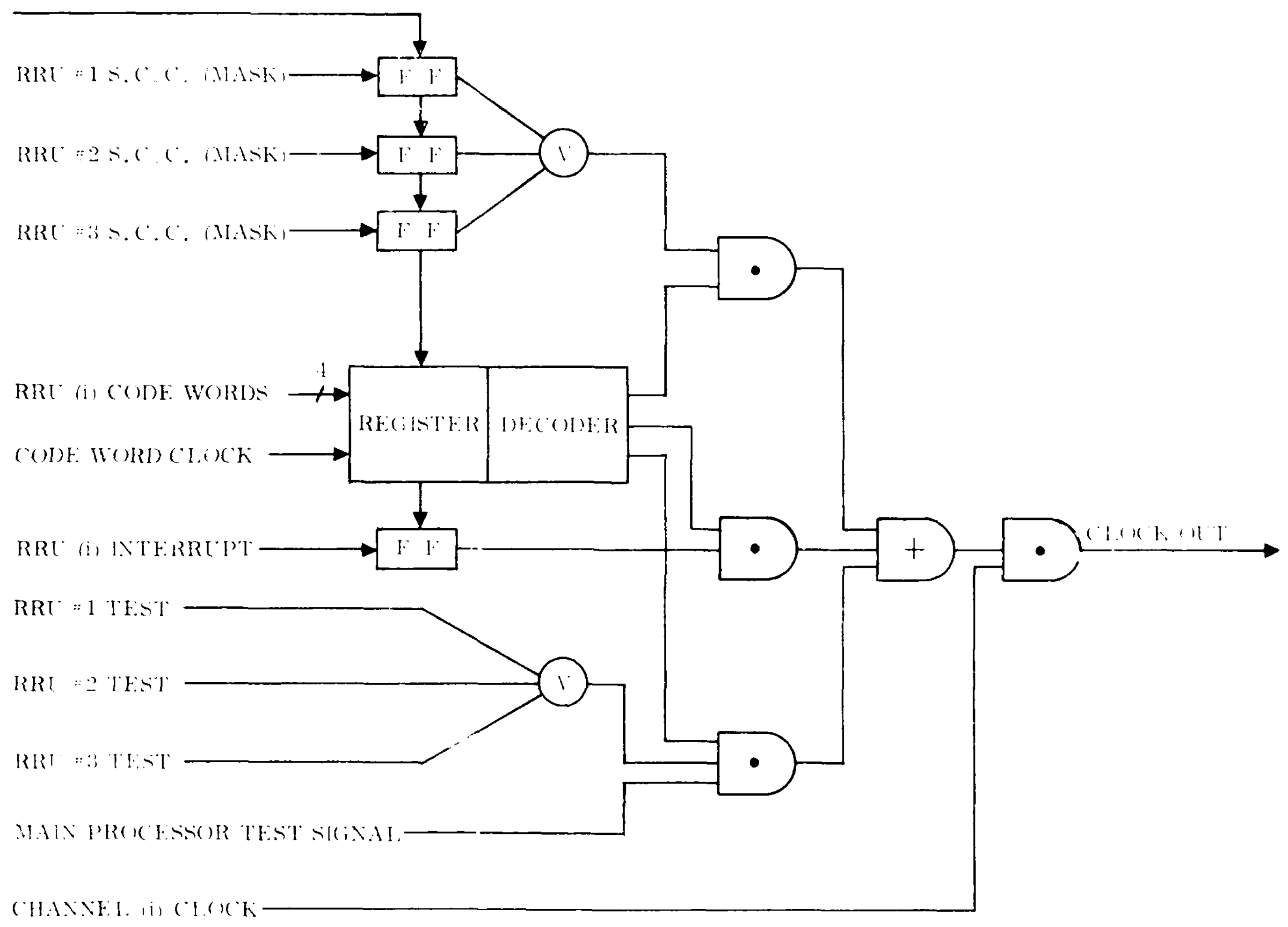


Figure 45. Single Channel RRU Clock Controller

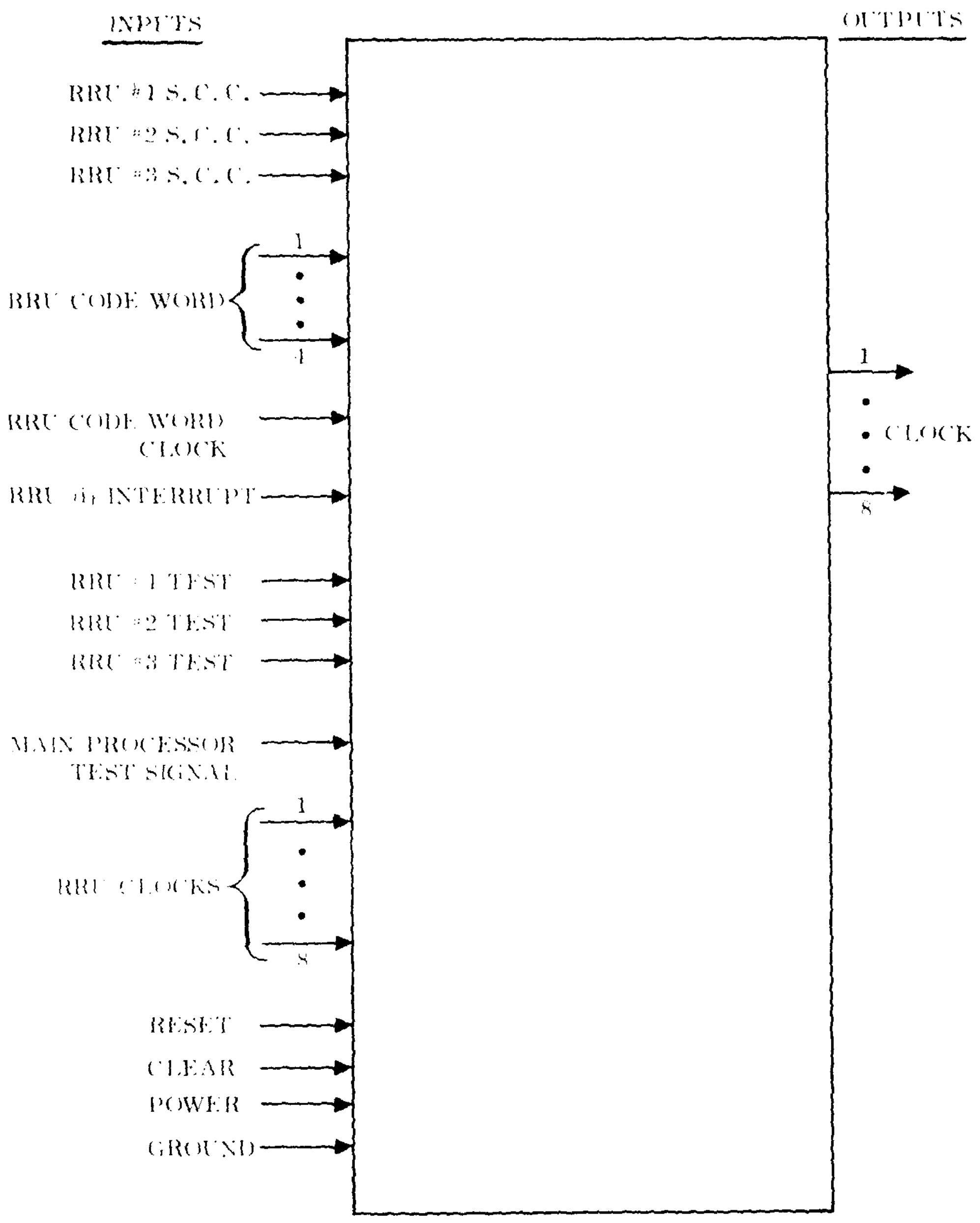


Figure 46. RRU Clock Controller Pin Assignment

## 6. Microsequencer

The proposed microsequencer chip is intended for use in the SDFTP microsequencer partition, but it can perform the microprogram sequence control for any similar microprogrammed processor. In the SDFTP it reduces the parts count and number of devices significantly because of the triplication of the microsequencer partition. In this partition it replaces four high-speed counter devices, two quadruple two-to-one multiplexers, and an eight-to-one multiplexer.

The microsequencer extends the microprogram address field width and adds a microprogram subroutine stack. The added capability is gained without slowing the sequencer or adding excessive power dissipation. The commercially available microsequencers generally offer more sophistication than required for a computer microprogram sequencer, with the attendant costs in power, speed, and reliability. They suffer from the need to address the largest possible market.

The prosposed microsequencer contains a microprogram register and incrementer, a two-word stack for microprogram subroutining, a tally counter for microprogram loop control, and a condition multiplexer to select jump tests from six data dependent conditions. A block diagram of the sequencer is shown in Figure 47. The function of the circuit is to generate the next microprogram address, given the current state of the machine. The possible next address, given that the current address n, is the next consecutive address $n + 1$, a jump address input from the control store, the address at the top of the subroutine stack (return from microsubroutine), or the fixed address zero used for honoring a program interrupt. The choice of the next address is controlled by the instruction input to the sequence, Table III, the external condition inputs, the tally counter, and the address input. The tally counter is loaded from the address input; it can be decremented, and generates a zero tally condition.

The function of the sequencer is as follows. The instruction decode logic, Table III, generates the control signals to operate the remainder of the sequencer. In addition to the instruction input ($I_0$ - $I_2$), the logic uses the output of the condition multiplexer, Figure 48, and the condition select inputs ($CS_0$ - $CS_2$). The tally counter Figure 49 is simply commanded to load from the address input, to decrement, or do nothing. It is a 12-bit 1's complement counter that uses the carry out as a zero detect. The counter is synchronous with carry look-ahead logic over a group of four bits. The heart of the sequencer is the multiplexer, microprogram register, and incrementer logic shown in Figure 50. The multiplexer chooses, on command, from the instruction decode logic, the input to the microprogram register from either the incrementer, the stack, the address in, or the constant zero. At each clock the microprogram register is loaded with the output of the multiplexer, which becomes the address output to the control store. The incrementer adds "1" to the microprogram register and makes the result available to the multiplexer and the stack.

The stack, Figure 51, is a two-word last in-first out stack. It is used to store the return address, n+1, when a microprogram subroutine is called. The two-word stack was selected because the control is particularly simple and the logic implementation is fast. Also, two-level subroutine

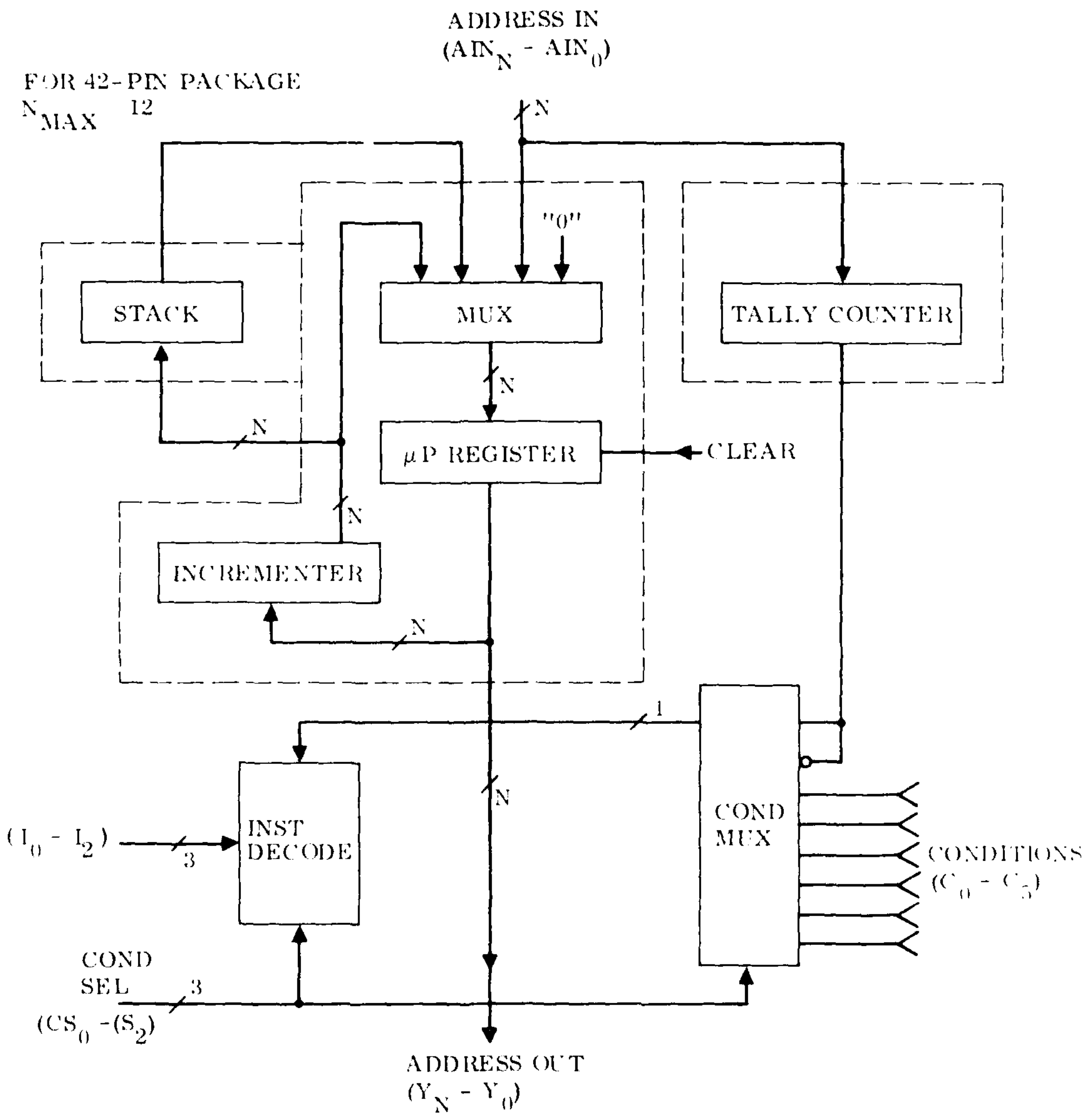


Figure 47. Microsequencer

## TABLE III. SEQUENCE TABLE

| OPCODE | OPERATION |
|---|---|
| 0 | BRANCH IF CONDITION TRUE |
| 1 | CALL IF CONDITION TRUE |
| 2 | RETURN IF CONDITION TRUE |
| 3 | LOAD TALLY |
| 4 | BRANCH IF CONDITION FALSE |
| 5 | CALL IF CONDITION FALSE |
| 6 | RETURN IF CONDITION FALSE |
| 7 | DECREMENT TALLY & BRANCH IF CONDITION FALSE |

INSTRUCTION DECODE EQUATIONS

| | | |
|---|---|---|
| INC ADDR | $\overline{\text{CLEAR}} \wedge \overline{\text{PUSH}} \wedge \overline{\text{POP}}$ | 1 |
| BRANCH | $\overline{\text{CLEAR}} \wedge (\overline{I}_0 \wedge \overline{I}_1 \wedge I_0 \wedge I_1) \wedge (I_2 \veebar X) \vee \text{PUSH})$ | - |
| PUSH | $\overline{\text{CLEAR}} \wedge I_0 \wedge \overline{I}_1 \wedge (I_2 \veebar X)$ | 1 |
| POP | $\overline{\text{CLEAR}} \wedge \overline{I}_0 \wedge I_1 \wedge (I_2 \veebar X)$ | 1 |
| LOAD TALLY = | $I_0 \wedge I_1 \wedge \overline{I}_2$ | 1 |
| TALLY COUNT = | $I_0 \wedge I_1 \wedge I_2$ | 1 |
| | | 15 GATES |

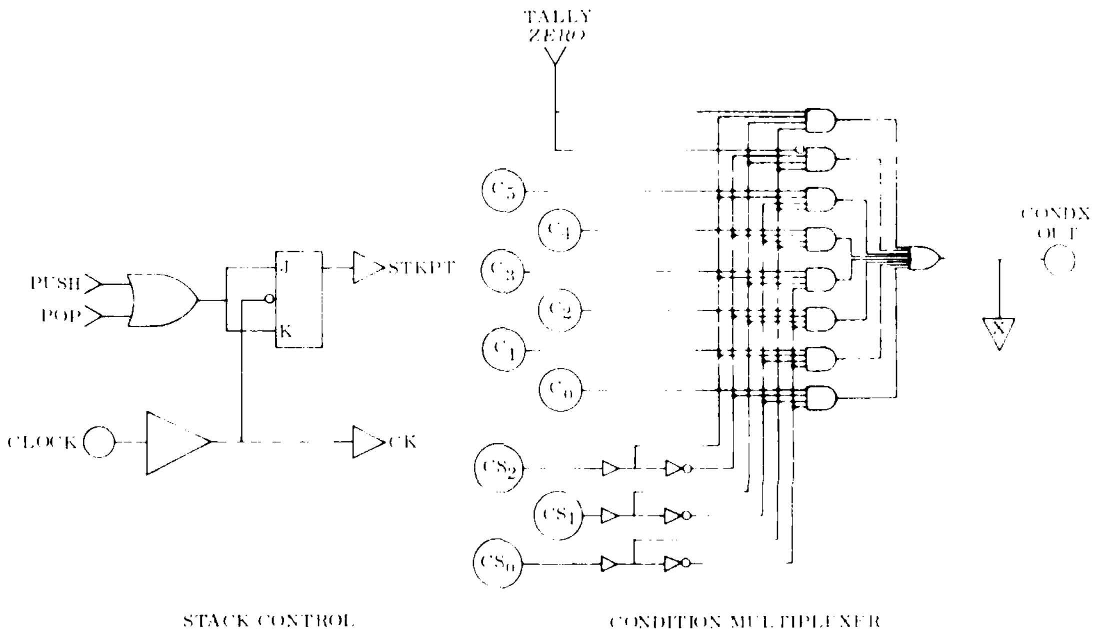
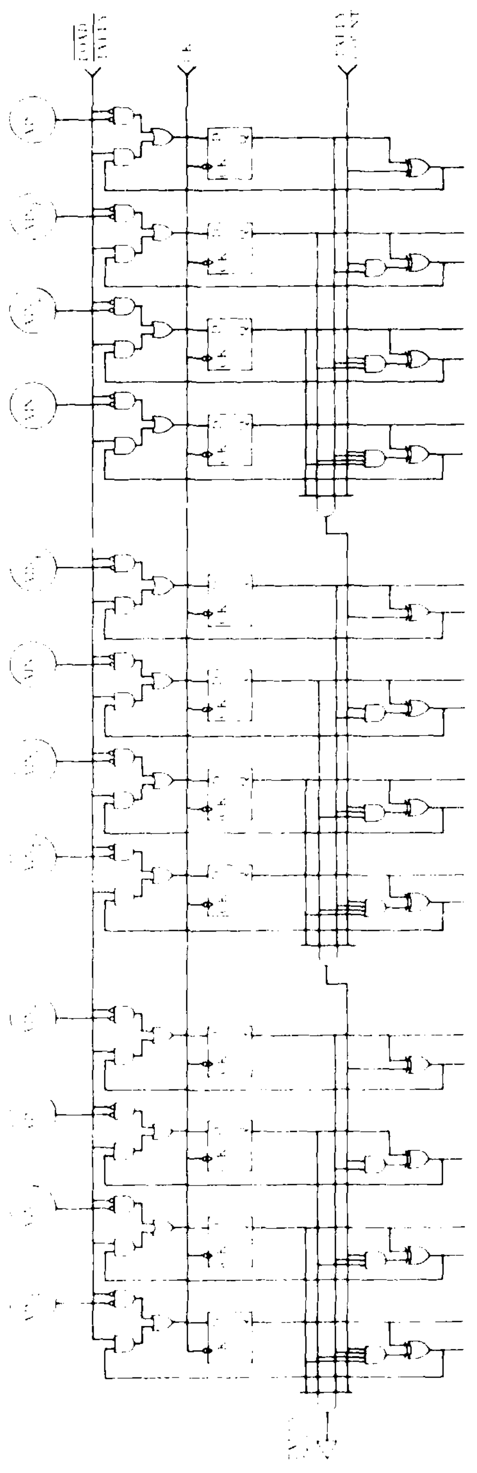


Figure 48. Microsequencer Control



Figuae 49. Tally Counter

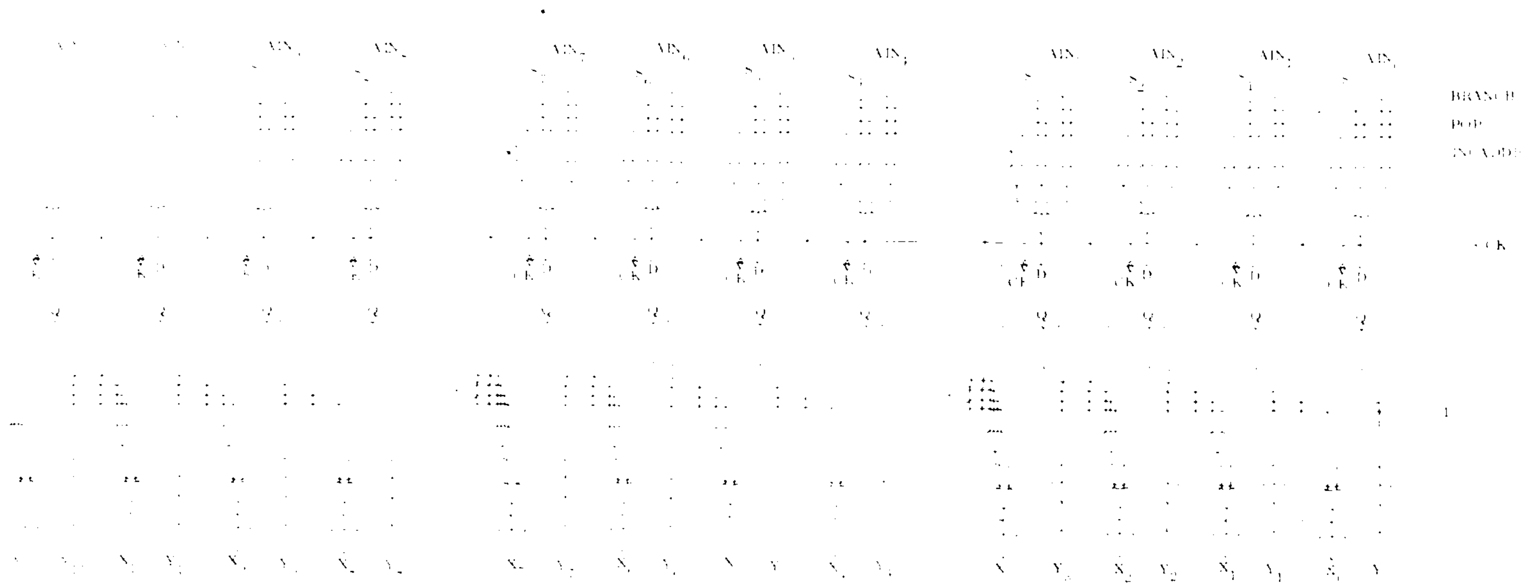


Figure 50. Multiplexer, Microprocessor Register and Incrementer

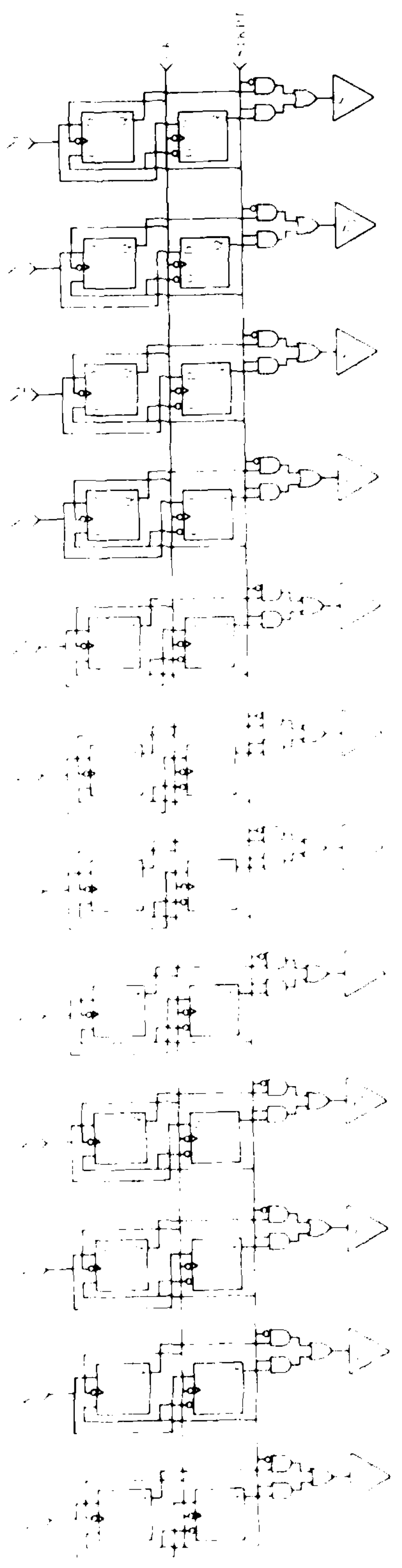


Figure 51. Microsequencer Stack

nesting is adequate for efficient microprogramming of almost all types of computer instruction sets. In fact, there is seldom a need for more than one level of subroutine. Larger stacks are often advocated when, the application program is part of the microprogram, which is not of interest here.

The above logic diagrams specify the function of the microsequencer chip. The approximate gate counts associated with each function are as follows:

| | | |
|---|---|---|
| Mux. Microprogram Register, and Incrementer | - | 179 |
| Tally Counter | - | 144 |
| Stack | - | 180 |
| Instruction Decode, Condition Mux. | - | 35 |

These total 528 gates, which is a modest-sized chip, especially when it is considered that many of the gates are part of multiplexers that can be realized by a few transistors in some technologies, and when it is noted that most structures used are quite regular.

## IV. SELF TEST

Self-test is used for a number of different reasons in this design. In the bit-slice processor, it is utilized in the on-line mode to determine whether one of two identical circuit implementations is faulted and, if so, which one. It is also used to periodically exercise the error detecting and reconfiguration circuits to verify that they are still unfailed. In the RRU, the Maskable S.C.C. trees are similarly exercised to, initially, detect errors; but if an error is detected it is used for locating the source of the error. The RRU computers are also self-tested to verify an unfailed status.

In the case of error detection, the self-test verifies that the coverage is in place and operable. In the error location mode, the self-test programs are not only executed to detect faults but, on completion of the tests, the precomputed final result can be compared with a pair of results generated by the SDFTP circuits to determine which circuit is in error. It is this latter usage that is important for diagnosing second errors in the bit-slice processor microsequencer and processor array partition, and in the RRU computers. (The control store second partition errors are resolved by table lookup in the RRU as discussed in Section III-F.)

To obtain a more confident estimate of the size of these programs and their effectiveness, particularly for circuit modules containing LSI devices, a 2901 bit-slice microprocessor device was simulated using the Digitest Version-4 Logic Automated Stimulus and Response (D4LASAR) facility and program system.

The D4LASAR program automatically generates high-quality diagnostic tests for complex sequential and combinatorial networks. When applied to an available gate level description of the 2901, the program determined that 3600 test vectors were required to obtain 99% fault coverage.

Thus, it is likely that a considerable number of vectors will be required to test circuits containing embedded LSI devices, such as the 2901, for this level of coverage. However, lower levels of fault coverage appear acceptable, based on the reliability predictions determined in Section VI. Thus acceptable self-test should be of manageable size if advantage is taken of additional factors. One of these is to use the information captured at the time of the error to localize the error and run those self-test segments that exercise those parts of the circuit contributing to these errors.

The self-diagnosing processor demonstration described in the Program Plan (Section VII), offers an opportunity to better quantify the size and coverage of these self-test programs.

# V. RELIABILITY ENHANCEMENT AND PREDICTION

## A. INTRODUCTION

The reliability enhancement of the SDFTP, with respect to the simplex processor, is achieved via static and dynamic redundancy. The bit-slice processor is partitioned to reduce the size of the circuit modules that are switched. This was accomplished without incurring undue interface switching costs. Hence, partition size is a compromise between the reliability improvement due to smaller partitions, which approach "component" reliability, and the loss of reliability due to additional interface devices, which are needed to interconnect the partitions and provide error detection and location information. The resulting three partitions have, roughly, equal failure rates as shown in Table IV with only four interfaces that must be monitored and controlled, as shown in Figure 3.

TABLE IV

PROCESSOR PARTITION FAILURE RATES

| | |
|---|---|
| MICROSEQUENCER | 5.66 |
| CONTROL STORE | 7.75 |
| PROCESSOR ARRAY | 7.73 |

Self-diagnosis is achieved through the use of self-checking checkers that extend the error protection boundary to include not only the partitions but the checkers themselves. These checkers are colocated at each interface with the devices that provide the interconnection between the partitions. These checkers monitor the partition outputs and alert the Reconfiguration and Recovery Unit of the occurrence of an error as soon as one occurs. These signals limit error propagation to the partition in which the error occurs by inhibiting the clock. Thus the bit-slice processor error state is maintained until the RRU computer can use the checker snapshots to locate the partition interface that is reporting an error.

After the error detection and location functions have been accomplished, the RRU reconfigures the bit-slice processor using the checker error information and the SDFTP status that it maintains. The RRU achieves the reconfiguration by emitting reconfiguration commands to the Voter-Switches located at each interface. These commands are intended to select either the voter or the switch connection between adjacent partitions.

At the start of each mission, the processor is configured with the voters providing the connection between the partition. After two errors are detected at an interface, the RRU commands the Voter-Switch to change over to the switch connection. Once reconfiguration has been completed, the RRU initiates the recovery process in the SDFTP in one of two ways: either the RRU vectors the bit-slice processor back to the last roll-back point or it vectors the processor to the self-test routine, which falls through to the rollback point in the application program if no errors are detected during the self-test exercise.

The calculation of the reliability of the SDFTP depends on the model of the design developed and the failure rates of the devices employed in the design. The model of the SDFTP design is described in the next unit. The discussion of the reliability estimate is concluded with a discussion of the use of the MIL-217B Handbook method used to calculate the failure rates.

## B. SDFTP RELIABILITY MODEL

The SDFTP reliability model consists of the serial reliability of the bit-slice partitions and the RRU. Each partition's reliability module is modeled as the cascade of the following devices.

1) an input S.C.C.
2) bit-slice processor partition
3) an output S.C.C.
4) Voter-Switch.

Since the bit-slice partitions are triplicated, the partition reliability expression is of the following form:

$$\text{Partition Reliability} = R^3 + 3R^2\ (1-R) + 3R\ (1-R)^2 \qquad (1)$$

where R is the reliability of a single partition.

The reliability of the input and output S.C.C. and the Voter-Switch voter, switch, combiner and command register are of a similar form. All but the voter and switch are considered to be in series in the reliability model. Because of the switching from the voter connection to the switch after the second error detection, the voter reliability modifies only the first two terms of the partition reliability expression, Eq. 1, while the switch modifies the last term of the expression. Hence, the reliability equation for the partition module is:

$$\begin{aligned}\text{Module Reliability} = & \Big[\left(R_p^{\,3} + 3\,R_p^2(1-R_p)\right)\left(R_v^{\,3} + 3\,R_v^{\,2}(1-R_v) + 3\,R_v(1-R_v)^2\right. \\ & \left.+ 3\,R_p(1-R_p)^2\right)\left(R_s^{\,3} + 3\,R_s^{\,2}(1-R_s) + 3\,R_s(1-R_s)^2\right)\Big] \\ & \left|R_{IC}^3 + 3\,R_{IC}^2\,(1-R_{IC}) + 3\,R_{IC}(1-R_{IC})^2\right| \cdot \\ & \left|R_{OC}^3 + 3\,R_{OC}^2\,(1-R_{OC}) + 3\,R_{OC}(1-R_{OC})^2\right| \cdot \\ & \left|R_C^3 + 3\,R_C^2\,(1-R_C) + 3\,R_C(1-R_C)^2\right| \cdot \qquad (2) \\ & \left|R_{CR}^3 + 3\,R_{CR}^2(1-R_{CR}) + 3\,R_{CR}(1-R_{CR})^2\right|\end{aligned}$$

where the reliability for the devices is

| | |
|---|---|
| $R_p$ | - *partition* |
| $R_v$ | - voter of the Voter-Switch device |
| $R_s$ | - switch of the Voter-Switch device |
| $R_{IC}$ | - input S.C.C. |
| $R_{OC}$ | - output S.C.C. |
| $R_C$ | - combiner of the Voter-Switch |
| $R_{CR}$ | - command register of the Voter-Switch. |

The microsequencer and control store-pipeline register follow this form exactly. However, the processor array requires that Eq. 2 be modified to account for the fact that it has three interfaces -- one input and two output The modification consists of multiplying Eq. 2 by the appropriate third interface devices. Since the processor array-microsequencer interface is identical to that modeled in Eq. 2, the processor-array-memory interface will be considered as the added interface. Thus, the first and second terms of Eq. 1 are now modified by two voter expressions. One is the processor array-microsequencer interface Voter-Switch device and the other is the processor array-memory interface device. The third term of Eq. 1 is modified by the two switch reliabilities of the two processor array output interfaces. Thus this part of the expression becomes,

P.A. Partition with voter and switch }
$$\left(R_p^3 + 3R_p^2(1-R_p)\right)\left(R_{vm}^3 + 3R_{vm}^2(1-R_{vm})\right) \qquad (3)$$
$$\cdot\left(R_{vo}^3 + 3R_{vo}^2(1-R_{vo})\right)$$
$$+ 3R_p(1-R_p)^2\left((R_{sm}^3 + 3R_{sm}^2(1-R_{sm}) + 3R_{sm}(1-R_{sm})^2\right)$$
$$\cdot\left(R_{so}^3 + 3R_{so}^2(1-R_{so}) + 3R_{so}(1-R_{so})^2\right) .$$

where

| | |
|---|---|
| $R_p$ | - processor array partition reliability |
| $R_{vm}$ | - processor array-microsequencer Voter-Switch voter |
| $R_{vo}$ | - processor array-memory Voter-Switch voter |
| $R_{sm}$ | - processor-array-microsequencer Voter-Switch switch |
| $R_{so}$ | - processor array-memory Voter Switch switch |

Equation 3 is multiplied by the reliability of the input S.C.C., the processor-array microsequencer interface S.C.C., and the processor-array-memory interface S.C.C., the Voter-Switch combiner and the command register reliabilities. Each of these circuits is triplicated and therefore modifies Eq. 2 by a factor of the form of Eq. 1. Eq. 4 is the processor array reliability expression:

$$
\begin{aligned}
\text{P.A. Partition} \quad & \Big[ \left(R_p^3 + 3R_p^2(1-R_p)\right)\left(R_{vm}^3 + 3R_{vm}^2(1-R_{vm}) + 3R_{vm}(1-R_{vm})^2\right) \\
& \cdot \left(R_{vo}^3 + 3R_{vo}^2(1-R_{vo}) + 3R_o(1-R_{vo})^2\right) \\
& + 3R_p(1-R_p)^2 \left(R_{sm}^3 + 3R_{sm}^2(1-R_{sm}) + 3R_{sm}(1-R_{sm})^2\right) \\
& \cdot \left(R_{so}^3 + 3R_{so}^2(1-R_{so}) + 3R_{so}(1-R_{so})^2\right) \Big] \\
& \cdot \left[R_{IC}^3 + 3R_{IC}^2(1-R_{IC}) + 3R_{IC}(1-R_{IC})^2\right] \\
& \cdot \left[R_{OCM}^3 + 3R_{OCM}^2(1-R_{OCM}) + 3R_{OCM}(1-R_{OCM})^2\right] \qquad (4) \\
& \cdot \left[R_{OCO}^3 + 3R_{OCO}^2(1-R_{OCO}) + 3R_{OCO}(1-R_{OCO})^2\right] \\
& \cdot \left[R_{CM}^3 + 3R_{CM}^2(1-R_{CM}) + 3R_{CM}(1-R_{CM})^2\right] \\
& \cdot \left[R_{CO}^3 + 3R_{CO}^2(1-R_{CO}) + 3R_{CO}(1-R_{CO})^2\right] \\
& \cdot \left[R_{CMM}^3 + 3R_{CMM}^2(1-R_{CMM}) + 3R_{CMM}(1-R_{CMM})^2\right] \\
& \cdot \left[R_{CMO}^3 + 3R_{CMO}^2(1-R_{CMO}) + 3R_{CMO}(1-R_{CMO})^2\right]
\end{aligned}
$$

where

$R_p, R_{vm}, R_{vo}, R_{sm}$, and $R_{so}$ are defined as before and

| | |
|---|---|
| $R_{IC}$ | - input S.C.C. reliability |
| $R_{OCM}$ | - output S.C.C. processor array-microsequencer interface |
| $R_{OCO}$ | - output S.C.C. processor array memory interface |
| $R_{CM}$ | - processor array-microsequencer Voter-Switch combiner |
| $R_{CO}$ | - processor array-memory Voter-Switch combiner |
| $R_{CMM}$ | - processor-array-microsequencer Voter Switch command register |
| $R_{CMO}$ | - processor array-memory Voter Switch command register. |

The processor array and the microsequencer partitions must be modified to account for the use of self-test to diagnosis which of two partitions has failed after two have been detected that affect the same partition. Since the coverage of these self-test programs is not complete, the term that represents the condition of two partitions failed and one unfailed is modified by a factor that accounts for this incomplete detection capability. This coverage factor is the conditional probability that, given that an error has occurred, the error is detected. For the microsequencer this term of Eq. 2 becomes,

$$3R_p(1-R_p)^2\left(R_s^3+3R_s^2(1-R_s)+3R_s(1-R_s)^2\right)CF_s\,, \tag{5}$$

where the terms are as defined for Eq. 2 and

$CF_s$ is the microsequencer self-test coverage probability.

For the processor array module, the corresponding term from Eq. 4 is,

$$3R_p(1-R_p)^2\left(R_{sm}^3+3R_{sm}^2(1-R_{sm})+3R_{sm}(1-R_{sm})^2\right)$$
$$\times\left(R_{so}^3+3R_{so}^2(1-R_{so})+3R_{so}(1-R_{so})^2\right)CF_{PA}\,, \tag{6}$$

where the terms are defined as in Eq. 4 and $CF_{PA}$ is the processor-array self-test coverage probability.

The reliability of the three partitions of the SDFTP is then the product of each of the partition modules reliability as given in Eq. 7,

$$\begin{array}{l}\text{Bit Slice}\\ \text{Processor}\\ \text{Reliability}\end{array} = R_m \times R_{cs} \times R_{PA}\,, \tag{7}$$

where $R_m$ - microsequencer partition module reliability

$R_{cs}$ - control store partition module reliability

$R_{PA}$ - processor array partition module reliability

The overall reliability of the SDFTP is the serial reliability of the bit-slice processor partitions and the RRU, which includes the Maskable S.C.C., and the RRU computer, together with the clock controller. Since there are three strings in the RRU, the reliability expression for the RRU is:

$$\text{RRU Reliability} = \left(R_T\cdot R_{RRC}\cdot R_{CC}\right)^3 + 3\left(R_T\cdot R_{RRC}\cdot R_{CC}\right)^2\left(1-R_T R_{RRC} R_{CC}\right)$$
$$+\ 3R_T R_{RRC} R_{CC}\left(1-R_T R_{RRC}\cdot R_{CC}\right)^2 \tag{8}$$

where $R_T$ is the reliability of the Maskable S.C.C. tree

$R_{RRC}$ is the reliability of the RRU computer including its memory/I/O buffers

$R_{CC}$ is the reliability of the clock controller.

The SDFTP reliability is then,

$$\text{Reliability}_{\text{SDFTP}} = R_m \times R_{CS} \times R_{PA} \times R_{RRU} , \tag{9}$$

where $R_{RRU}$ is the reliability of the RRU.

Once the reliability model has been established, the reliability of the design can be calculated using the device failure rates.

## C. FAILURE RATE CALCULATIONS

The reliability assessment performed on this program follows that given in Military Standardization Handbook MIL-HDBK-217B, Reliability Prediction of Electronic Equipment. Device failure rates were determined using the expressions given there for monolithic solid state integrated circuit devices as,

$$\lambda_p = \pi_L \pi_Q (C_1 \pi_T + C_2 \pi_E) , \tag{10}$$

where $\lambda_p$ - the device failure rate in failures/$10^6$ hours

$\pi_L$ - is the device learning factor

For this projection, $\pi_L$ was assumed to be 1 for all devices (in production)

$\pi_Q$ is the quality factor

$\pi_Q$ was assumed to be 2 corresponding to quality level B, MIL-M-38510, Class B (JAN)

$\pi_T$ - is the temperature acceleration factor

An average junction temperature of 75°C resulting in $\pi_T$ of 1.6.

$\pi_E$ - is the application environment factor

This was selected as 6.0 corresponding to an airborne, uninhabited environment.

The complexity factors for the SSI, MSI logic and ROM memory were determined using the gate counts and tables given in the Handbook. For the LSI devices, where gate estimates were not available, manufacturer information was used where available and, where not available, a gate count was estimated based on the logic equivalents. Complexity factor projections for $C_1$ and $C_2$ were developed using the Handbook per gate values for the optimum level of integration. This had the effect of making the per gate complexity factors less sensitive to gate count for the device complexities of interest to this program, and reduced the effect of the gate count uncertainty of the highly-integrated LSI devices, such as the 8048 computer.

## D. RELIABILITY ESTIMATES

Using the device failure rates calculated as described above, the reliability of the simplex processor consisting of a microsequencer, control store and processor array, was calculated. The corresponding probability of failure for one to 10 hour missions were calculated and are shown in Figure 52. The SDFTP probability of failure for various coverage factors is plotted in Figure 53.

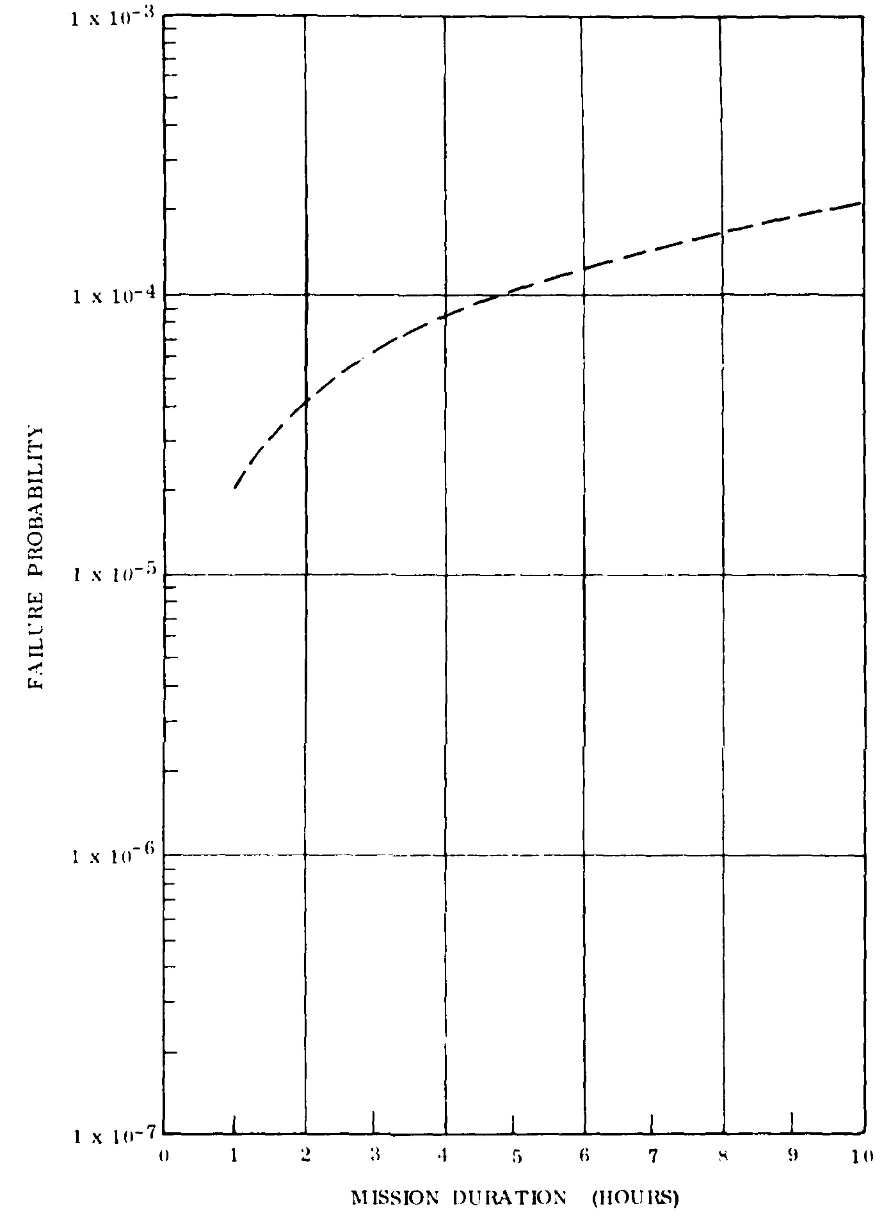


Figure 52. Simplex Processor Probability of Failure

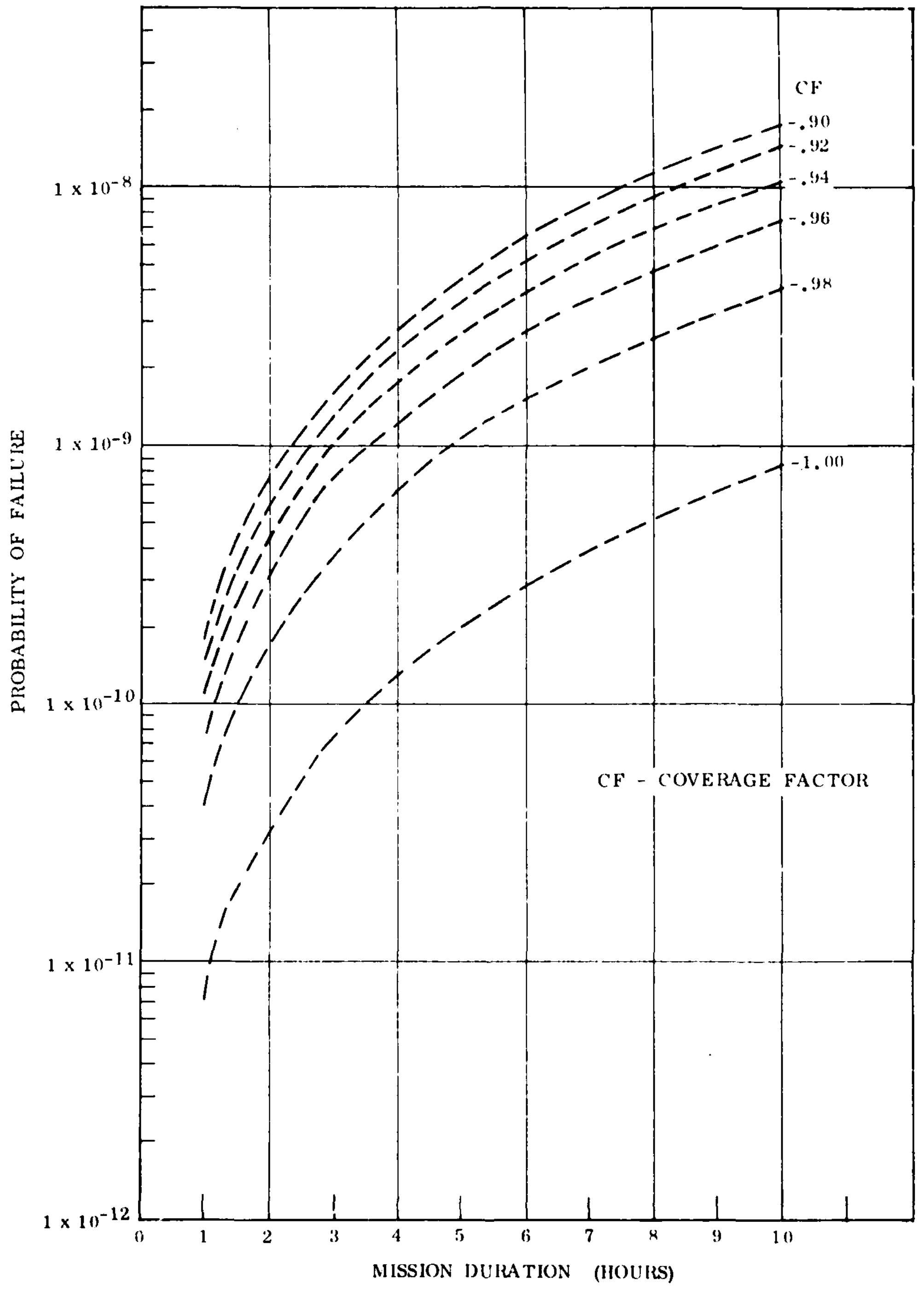


Figure 53. SDFTP Probability of Failure

## VI. COMPARISON OF SDFTP AND SIMPLEX PROCESSORS

The Self-Diagnosing Fault-Tolerant Processor (SDFTP) provides "failed $op^2$" fault tolerance. In comparison with the simplex design, which cannot tolerate a fault, it is significantly superior -- especially for fault tolerant applications such as flight control. For short missions, less than 10 hours, it has significantly enhanced reliability compared with the simplex design. Its failure rate for a two hours mission is nearly four orders of magnitude less than the simplex design, as described in Section V.

Testability and maintainability of the SDFTP is significantly improved over the simplex design since it maintains an error history and up-to-date status of the SDFTP during the entire flight, which can be utilized to decrease repair times. The incorporation of self-test programs, coupled with self-checking checkers and partition interface scanning registers, significantly increases the diagnosis capability since the error reporting is to a much finer scale than it is in the simplex design.

In the performance and ease of application the two designs are comparable. The rate of instruction execution of the SDFTP should approach that of the simplex design, provided that the processor control circuitry does not entail large delays. Since the instruction repertoires of the two designs are nearly the same, the ease of programming should be nearly the same. It is intended that the SDFTP have some additional instructions to ease the reconfiguration and recovery process.

Since the improved fault tolerance and reliability of the SDFTP is achieved via redundancy, the SDFTP requires much larger resources than the simplex processor. As listed in Table V, the simplex processor only requires 45 devices, (without the microsequencer device) while the SDFTP requires about 4.4 times as many devices. Most of the additional parts are required to implement the triplication employed with the bit-slice processors and the RRU error processing channels. The remainder are needed to implement the checkers, partition interconnection devices and the dynamic redundancy control. In terms of the number of different devices (parts count) the simplex processor requires just 12 different devices, all of which are commercially available. In contrast, the SDFTP is implemented with 19 different devices, of which four are special designs, as shown in Table VI. Both the simplex and the SDFTP would benefit from the special microsequencer device design, since it would reduce the simplex parts count by six and the SDFTP count by 18. The microsequencer device would also eliminate three different parts thereby reducing the parts types by two. The SDFTP is much more extensively integrated than the simplex, with nearly half of the devices of the LSI level of integration.

TABLE V

COMPONENT COMPARISON
OF
SIMPLEX VERSUS SDFTP

| FUNCTION | SIMPLEX | SDFTP |
|---|---|---|
| BIT-SLICE MICROPROCESSOR | | |
| LSI DEVICES | 16 | 3(16) |
| MSI DEVICES | 26 | 3(26) |
| SSI DEVICES | 3 | 3(3) |
| TOTAL | 45 | 135 |
| CHECKERS | | |
| S.C.C. (LSI) | | 21 |
| RECONFIGURATION | | |
| VOTER-SWITCH (LSI) | | 10 |
| RRU | | |
| MONOLITHIC MICROCOMPUTER (LSI) | | 27 |
| CLOCK CONTROLLER | | 3 |
| MISC. | | 12 |
| OVER-ALL TOTAL | | 208 |

TABLE VI

PARTS COUNT COMPARISON

SIMPLEX VERSUS SDFTP

| SIMPLEX | SDFTP |
|---|---|
| LS00 | LS00 |
| LS04 | LS04 |
| LS86 | LS86 |
| LS138 | LS138 |
| LS151 | LS151 |
| LS158 | LS158 |
| LS163 | LS163 |
| LS174 | LS174 |
| LS253 | LS253 |
| 2902 | 2902 |
| 2901 | 2901 |
| 5341 | 5341 |
|  | S.C.S. WITHOUT MASK |
|  | S.C.C. WITH MASK |
|  | VOTER-SWITCH |
|  | CLOCK-CONTROLLER |
|  | 8084 |
|  | 8243 |
|  | 8355 |

Storage requirements for the two designs are difficult to compare. The simplex processor has nothing equivalent to the RRU computer-storage devices. In the bit-slice processor area, the requirements are hard to quantify for the reasons cited in Section IV. However, it is believed that, because the SDFTP is partitioned into smaller circuit modules than the simplex processor, the SDFTP can use smaller diagnostic programs than the simplex processor, with higher coverage.

Another advantage of the SDFTP is the absence of a hard-core problem since two replicas should be operable under the single fault-at-a-time assumption.

# VII. PROGRAM PLAN

It is recommended that a prototype self-diagnosing fault tolerant processor be built to demonstrate the concepts and techniques involved, and to indicate the resources required for their implementation. This demonstration will indicate whether the self-diagnosing processor has the ability to operate correctly through the introduction of faults in the processor, and whether it can execute the prescribed tasks in a timely fashion. These experiments will be designed to demonstrate the following qualities:

- degree of tolerance
- comprehensiveness of protection
- responsiveness of error processing
- types of fault coverage
- compatibility with LSI implementation

The ability to protect the processor from both single and multiple errors, single and double fault occurrences of the same fault without intervening repair, and consistent and inconsistent types of errors will be demonstrated. The ease and variety of fault insertion are important attributes because they allow the effectiveness of the error detection, error location, reconfiguration, and recovery capabilities of the processor to be readily exhibited. Thus the ability to display the state and readiness of the demonstrator, as well as the fault history, are important considerations in developing an effective demonstrator presentation.

## A. DEMONSTRATOR DEVELOPMENT PLAN

The recommended approach consists of a two-phase development program. The objective of this program is to construct a demonstrator together with its associated demonstration programs, which will show the operation of the self-diagnosing processor under various conditions of fault. The first phase will be concerned with definition and design specifications, and the experiments that can be run on the demonstrator. The resulting definition will then be used to establish the requirements and specifications of the demonstrator. The self-diagnosing processor will be built during the second phase, according to the specifications arrived at in Phase I; the result will be a self-diagnosing processor and its associated computer programs to demonstrate the capabilities of the processor.

### 1. Definition, Design and Specification Phase

The following tasks shall be accomplished during this phase:

#### Define Demonstration System

A plan for demonstrating the capabilities of the Self-Diagnosing Fault Tolerant Processor (SDFTP), including the experiments that are to be performed to illustrate the features and range and extent of the fault tolerance desired by the Air Force, will be developed. A functional description of the Self-Diagnosing Fault Tolerant Demonstration System will be provided.

## Design

Using the Air Force approved SDFTD plan the functional specifications of the demonstrator, including processor design demonstration requirements and principal interfaces, will be established. This task will have two distinct parts, preliminary design and detailed design.

In the preliminary design work the Self-Diagnosing Design Techniques Demonstrator (SDFTD), including the self-diagnosing fault tolerant processor (SDFTP), will be designed to a level that clearly shows the technical adequacy of the selected approach and establishes the ability of the SDFTD to demonstrate the fault tolerance of the processor.

This preliminary design effort will result in a SDFTP hardware development specification that covers (1) all essential system functional characteristics, (2) necessary interface characteristics, (3) specific designation of the functional characteristics to key configuration items, and (4) tests that will verify that the specified performance has been achieved. A Computer Program Development (built to) Specification will also be written that describes in operational, functional, and mathematical language, all of the requirements necessary to design the required computer programs in terms of performance criteria.

In the detailed design portion the SDFTD will be designed to a level that clearly shows that all design requirements are satisfied, that the design is essentially complete, and that the fabrication drawings are ready for release. This work will culminate with a presentation of the detailed design to the Air Force for approval. The SDFTD detailed hardware requirements will also be developed during this part of the effort. These requirements shall be specified in a hardware functional description that establishes the performance, design and fabrication requirements. Design drawings will be provided to good commercial practice. The SDFTP computer programs will be described in a Computer Program Product Specification that provides a summary of the purpose and scope of the specification and a review of the major functions. The requirements section will provide for a functional allocation description, functional description, storage allocation, functional flow diagram, program interrupts, and control logic description.

## Program Plan

The Phase I report will include a plan for the implementation and demonstration of the self-diagnosing processor designed during Phase I. This plan will include a description and schedule of the major events in hardware and software construction, test and demonstration. Estimates of material cost, labor by type, and schedule will be included.

## 2. Fabrication, Test and Demonstration Check-Out Phase

Working from the detailed design information developed in Phase I. the demonstrator will be built and the associated computer programs will be written. The custom LSI devices that were designed during this development will be implemented in small-scale integrated, SSI, and medium-scale integrated, MSI, circuit form. These devices, as well as those needed to implement the remainder of the design, will be selected from commercially available products. Layout and fabrication of the custom LSI devices is planned following successful operation of the demonstrator.

Checkout of the hardware and the software are scheduled to proceed concurrently, using available development systems. Integration of the hardware and the software will be accomplished as the individual subsystems, programs and routines are tested and checked out. The completely integrated demonstrator will be tested to verify that the demonstrator performs as specified and the fault insertion experiments can be successfully performed.

The proposed schedule for the development of the demonstrator is shown in Figure 54. Significant milestones are also indicated. The definitive plan and schedule for this second program phase will be refined at the end of the Definition, Design and Specification phase as indicated in the Program Plan, above.

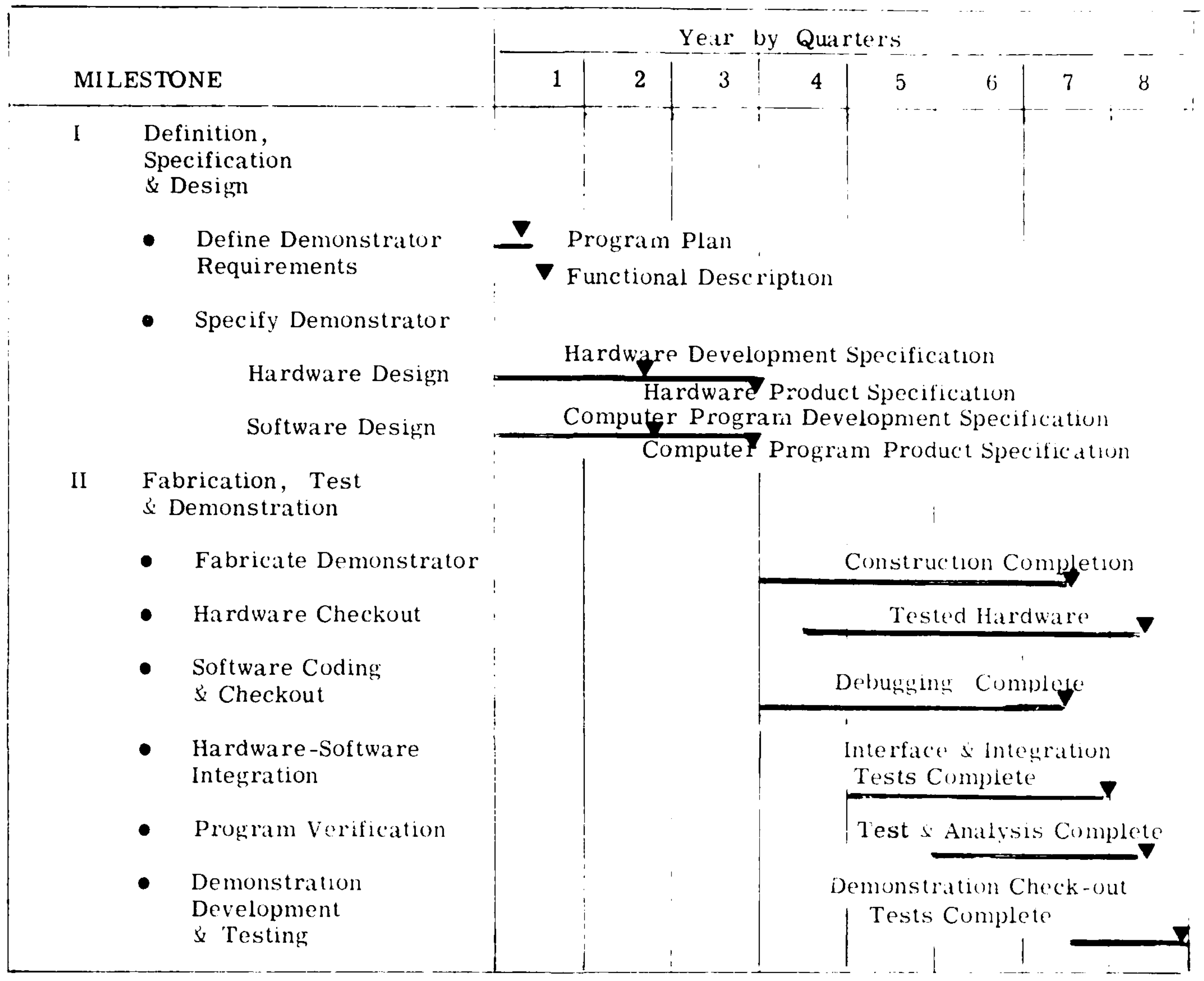


Figure 54. Proposed Schedule - Self-Diagnosing Design Techniques Demonstrator

## REFERENCES

1. D.A. Andersen, "Design of Self-Checking Digital Networks Using Coding Techniques", Coordinated Science Laboratory Report R-527, University of Illinois, September 1971

2. M. Diaz, "Design of Totally Self-Checking and Fail-Safe Sequential Machines", Digest of the Fourth Annual Symposium on Fault Tolerant Computing, pp. 3-19 to 3-24, June 1974.

3. R.W. Cook et al, "Design of a Self-Checking Microprogram Control", IEEE Trans. on Computers, Vol. C-22, pp 255-262, March 1973.

4. D.A. Anderson and G. Metze, "Design of Totally Self-Checking Check Circuits for m-out-of-n Codes", IEEE Trans. on Computers, Vol. C-22, pp 263-269, March 1973.

# APPENDIX I
# DEFINITION OF BASELINE PROCESSING REQUIREMENTS

## TABLE OF CONTENTS

## A. INTRODUCTION AND SUMMARY

Two airborne application areas were selected for the baseline requirements. These are airborn flight control and the synthetic aperture ground may function of airborne multimode radar signal processing. Each application is examined to determine its processor requirements, beginning with mission identification and functional analysis leading to the development of the algorithm flow. Performance analysis of representative tasks is described and the resource estimates are developed in terms of memory, processor speed, and complexity, as measured in terms of the variety of operations and their corresponding execution rates.

The Flight Control Application is considered first in Section B since it represents a set of requirements that falls within the realization capabilities of existing LSI devices, such as microprocessors and memories, configured in a single programmable computer structure. It is estimated that a high performance control-configured, fly-by-wire aircraft would require less than 16,000 words of 16-bit wide memory and could be controlled by a processor capable of executing instructions at a 300 to 400 thousands of operations per second (KOPS) rate. Because of the safety requirements of this application, quadruple redundancy coupled with software implemented redundancy management leads to a sophisticated input/output system that connects the electronic flight control system to the aircraft control sensors and actuators. The reconfiguration approach is designed to achieve "failed op-squared" fault tolerance for the electronics.

The second application, Synthetic Aperture Ground Map Processing of a Multimode Radar, described in Section C results in signal processor requirements that are beyond the capability of current and near future single conventional microprocessor designs. However, special programmable pipeline processors and netted sets of microprocessors are believed to be capable of achieving the performance required. As in many of the other radar signal processing applications, the core signal processing function has the ability to generate a doppler frequency analysis of the radar return. For this ground mapping mode of the multimode radar, a processing rate in excess of $20 \times 10^6$ complex multiplies is required in addition to a signal processing operation rate in the 1-2 million instructions per second range. Compared to the flight control application, the multimode memory requirements are significantly larger and are estimated to fall in the 3.5 million bit range. This storage is normally distributed throughout the signal processor and must provide a high memory accessing rate capability, which is a function of the specific radar mode and signal processor architecture.

B. AIRCRAFT FLIGHT CONTROL

This electronic flight control system combines contemporary ideas for reconfiguration (transient fault recovery, computer self-monitoring) with conventional hardware redundancy techiques in a basic quadruplex redundant structure. With appropriate operating software, the system provides the reliability and fault tolerance, which are typically characterized as "failed op-squared" performance. In addition, the system automatically recovers from certain transient faults, such as interruption of electrical power, and reorders itself to obtain the highest available level of redundant operation. A high performance control-configured vehicle (CCV), fly-by-wire (FBW) aircraft and control surfaces are shown in Figure I-1. Table I-1 is a summary of the major aircraft functions with respect to aircraft safety criticality In the following, we will be primarily interested in the flight crucial functions since they are performed in the flight control system.

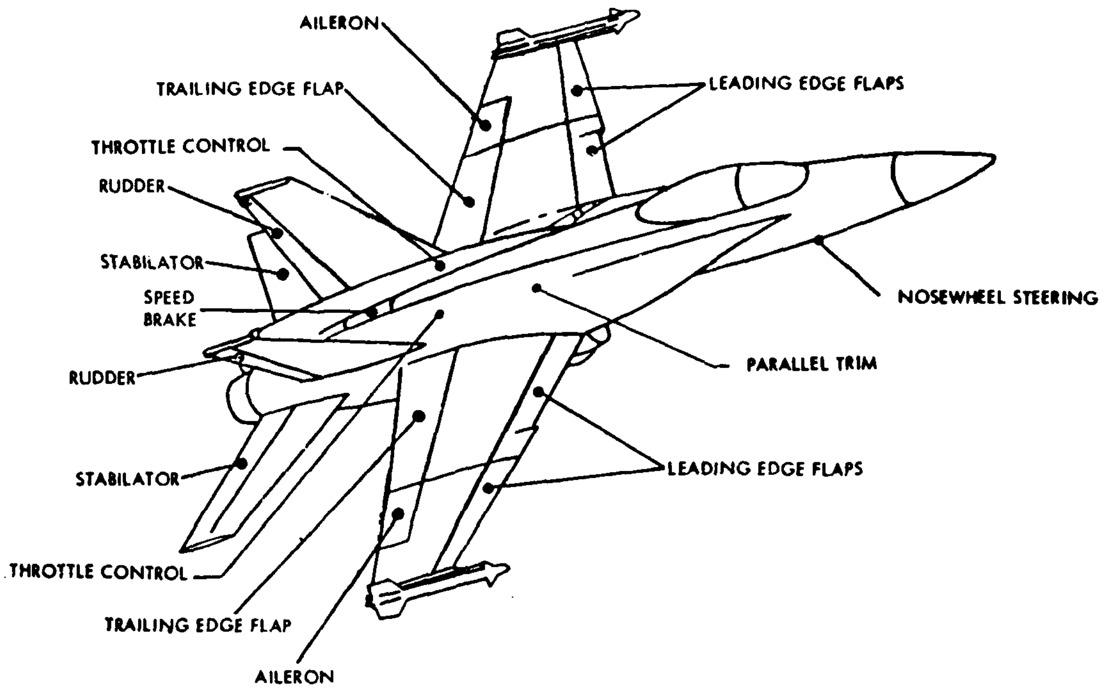


Figure I-1. Flight Control Electronic System Control Surfaces

Transformation of the operational criteria into design requirements for a fault tolerant (redundant) digital computer system is arranged to obviate single point system failures. The design must also meet the following fundamental design requirements:

. Each computer unit shall independently assess its, and the system's, operational status.

. No computer or combination of computers shall interrupt another computer's normal operation.

- The system's redundant operation must start up, and recover from transient fault conditions without flight crew intervention.
- The system design must be able to achieve functional operation down to a simplex string of operable elements and be architecturally expandable to at least quadruplex redundancy.

TABLE I - 1. FUNCTION SUMMARY FOR APPLICATION MODELS

| | |
|---|---|
| Flight Crucial Functions | . Flutter Suppression<br>. Structural Mode Suppression<br>. Fly-by-Wire Control<br>. Full-time Stability Augmentation |
| Flight Critical Functions | . Category III MLS Autoland |
| Noncritical Functions | . Track Angle Select Hold<br>. Flight Path Angle Select 'Hold<br>. 2D '3D 4D Command Generation<br>. Air-Ground Data Link for ATC Communication<br>(Above functions provided by Navy 'Guidance Computers) |

Proceeding from these requirements, a software implemented redundant management approach leads to the inclusion of a reconfiguration process consisting of failure isolation, transient fault recovery, and redundancy degradation. The redundant channel processes are consolidated at two system nodes: at the sensor signal input to the control law computations and at the servo actuator output. The sensor signal selection process is mechanized in software and the output voting node is a hydromechanical mechanization. However, the majority of the reconfiguration mechanisms are software processes designed to achieve system flexibility and adaptability. The hardware architecture, by virtue of its communication interconnections, is what makes it practically possible to achieve the benefits of reconfiguration.

The computer unit is replicated on a per channel basis to build a redundant (in this case, quadruplex) fault tolerant system. A processor and all channel interface electronics are included in the computer unit. Sensor, mode control and servo hardware interfaces are dedicated on a channel basis. All cross-channel communication is accomplished via dedicated one-way serial digital data buses that independently interconnect each computer to each other, providing complete electrical isolation between channels. Each computer exclusively controls the engagement and shutdown of its own servos. A block diagram of an integrated navigation/guidance/flight control system is shown in Figure I-2. The assignment of channels and input/output electronics to the computer units is shown in Figure I-3. A more detailed view of a single computer's sensor and actuator relationship is shown in Figure I-4.

The control surface functions are pictured in Figures A-1 through A-9 in Addendum A. They are:

. Stabilator Functions

. Trailing Edge Flap (TEF) Functions

. Leading Edge Flap Functions

. Rudder Functions (Channels 1 and 2)

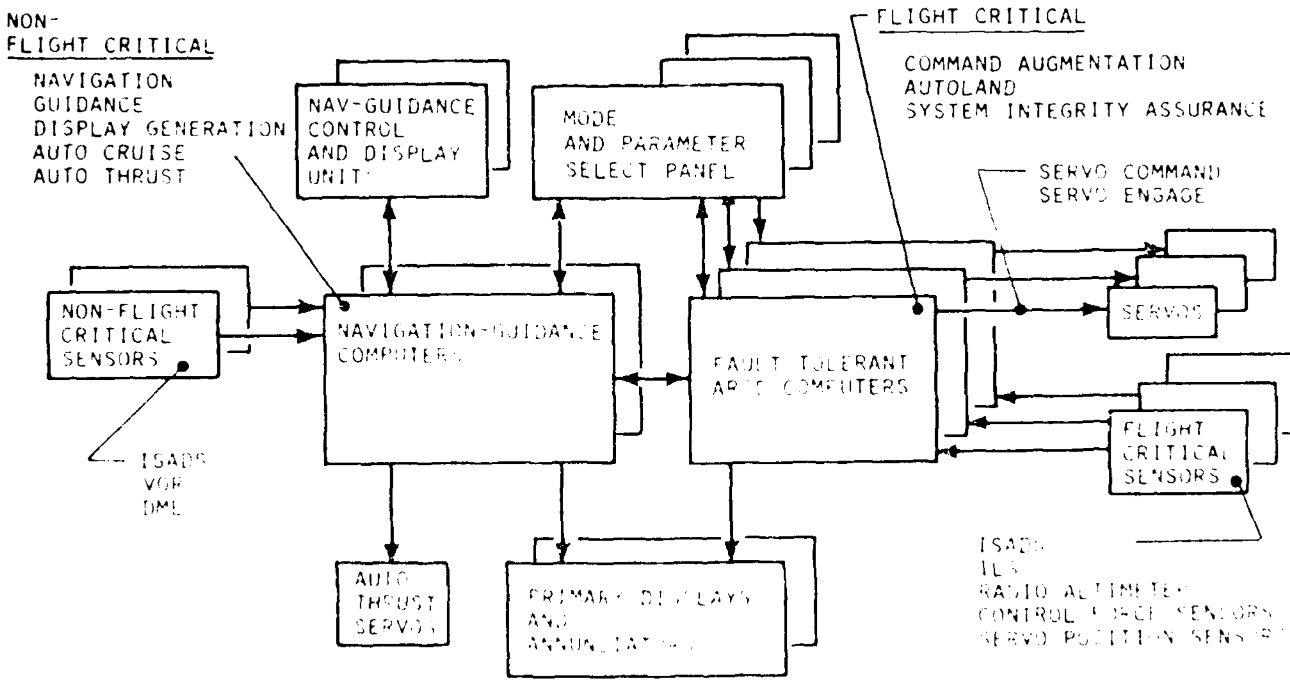


Figure I-2. Integrated Navigation/Guidance/Flight Control System

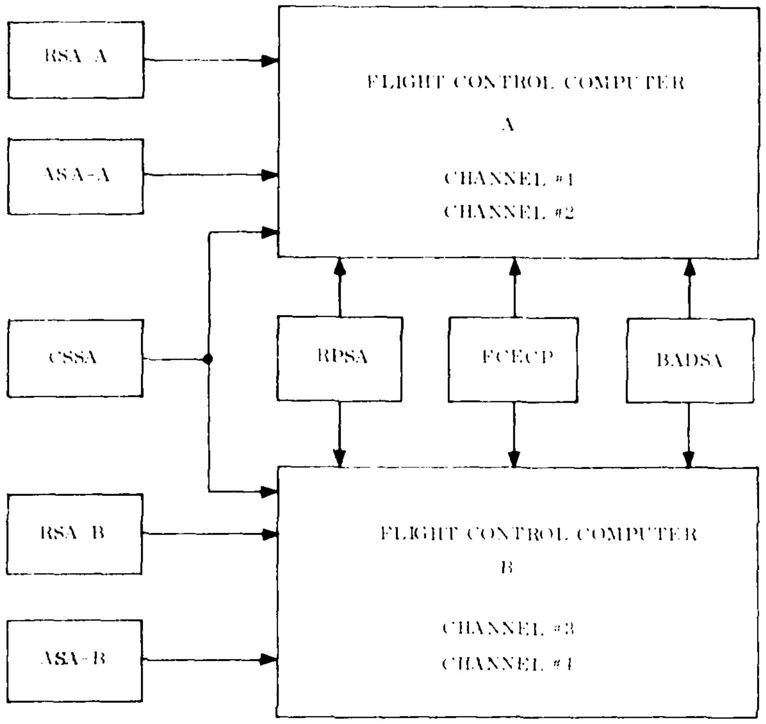


Figure I-3. Flight Control Electronics Set

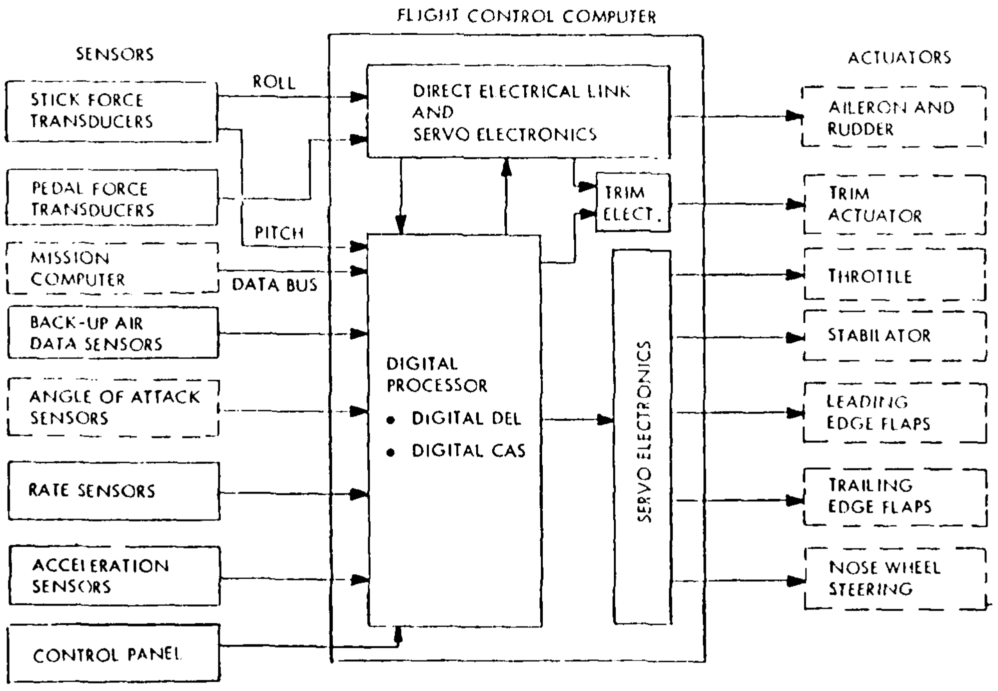


Figure I-4. Flight Control System

- Rudder Functions (Channels 3 and 4)
- Aileron Functions (Channels 1 and 2)
- Aileron Functions (Channels 3 and 4)
- Nose Wheel Steering Functions
- Approach Power Control Functions

The interfaces between the computer software programs and hardware and the rest of the flight control system are shown in Figure I-5.

The set of representative control laws for a CCV/FBW application are shown in Figures I-6 through I-17. An overview of the individual pitch, roll, yaw, flutter, and maneuver for autoland, go-around, and CCV/FBW is shown in Figure I-6. The individual control law diagrams are referenced in Figure I-6 and presented in Figures I-7 through I-15. Sensor and mode control interface requirements and servo and display interface requirements are shown in Figures I-16 and 17.

Processor resource estimates for a high-performance FBW aircraft flight control system designed to meet the foregoing requirements are tabulated in Table I-2. The total storage requirements are approximately 13,000 16-bit words of program storage and 1,300 16-bit words of data memory. The performance needed is about 320,000 operations per second. These estimates are obtained through sizing the application on a 16-bit flight control computer having the instruction repertoire and execution times shown in Addendum B.

## C. MULTIMODE RADAR SYNTHETIC APERTURE GROUND MAPPING

A multimode radar may have a number of modes of which the following four are typical:

1) Medium PRF Air-to-Air Search
2) High Resolution Spotlight Mode Synthetic Aperture Radar (SAR) Mapping
3) Non-cooperative Target Recognition (NCTR)
4) Terrain Following/Terrain Avoidance (TFTA)

The associated radar signal processor should be capable of not only processing each of the mode returns in real time but should be capable of switching between any pair of modes in real time without hardware changes.

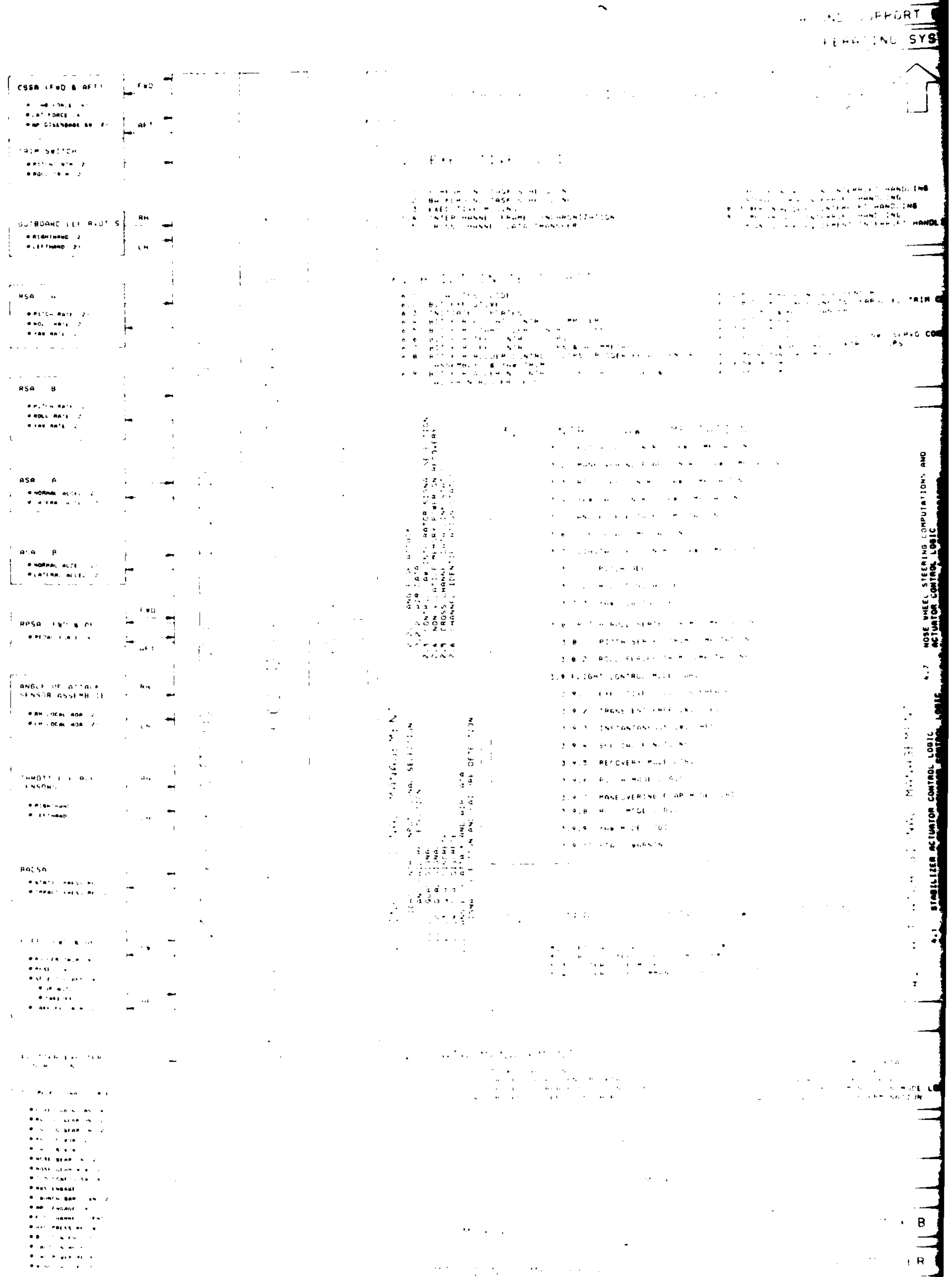
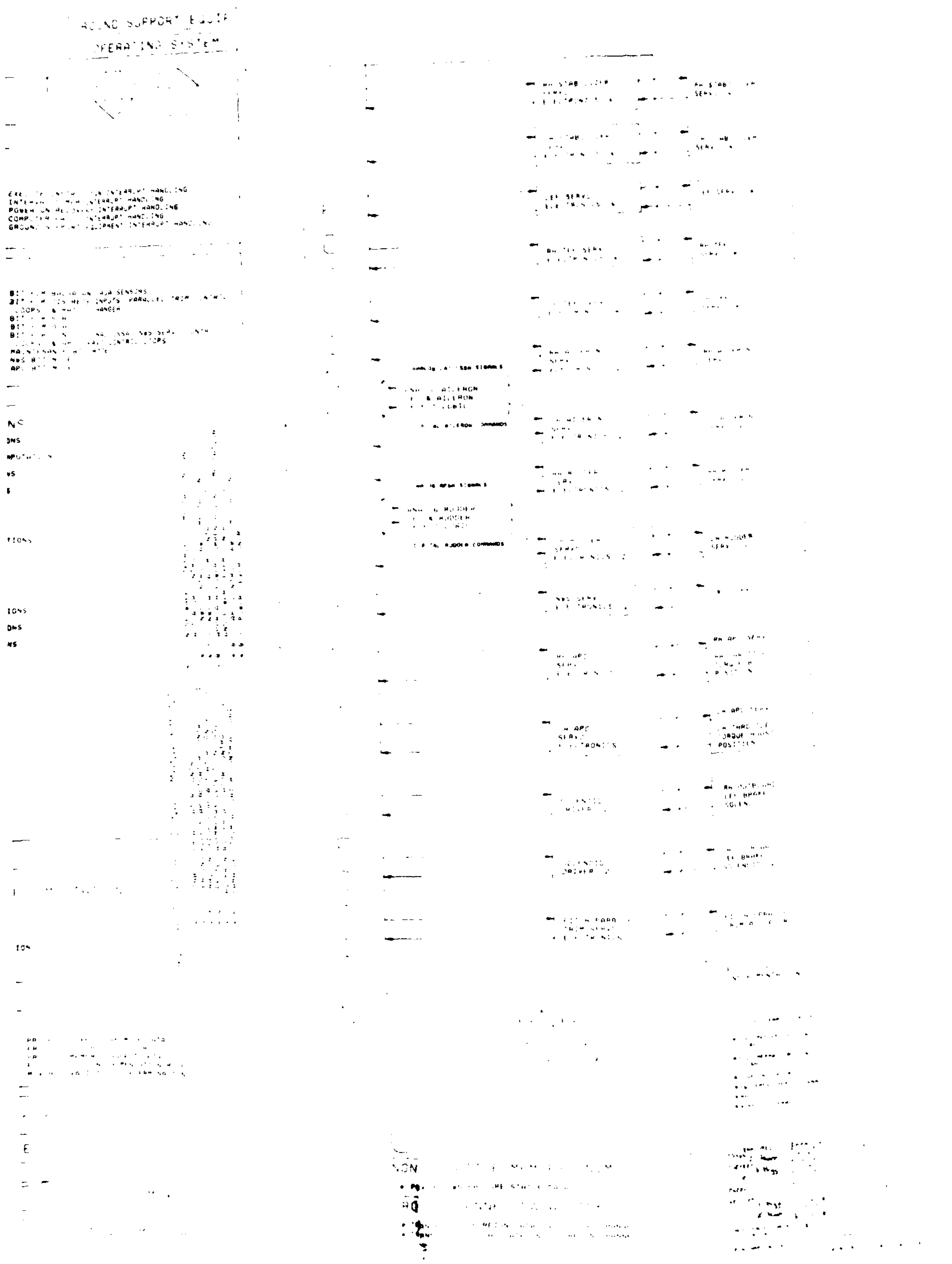


Figure I-5. OP Computer Interface Block Diagram.



Block Diagram.

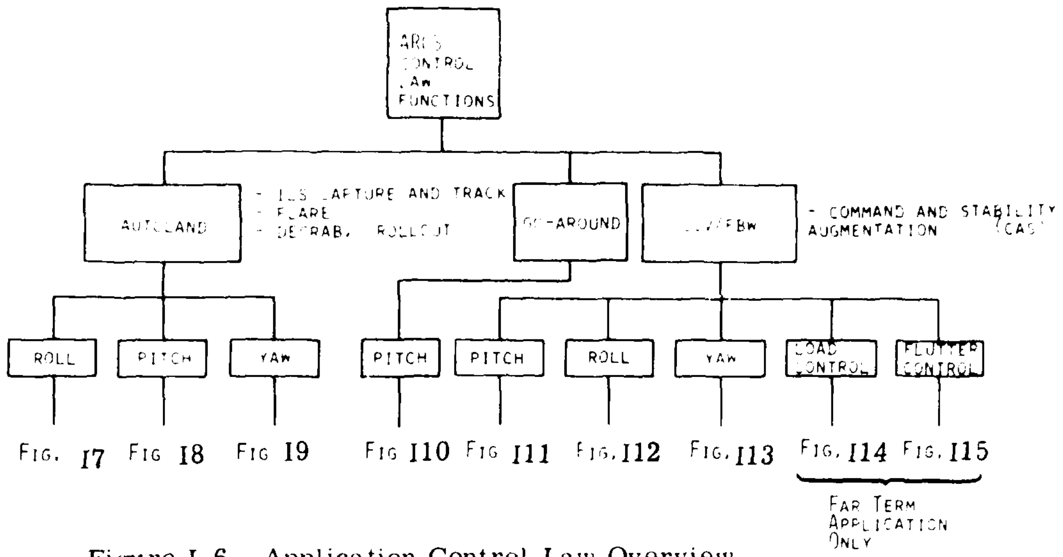


Figure I-6. Application Control Law Overview

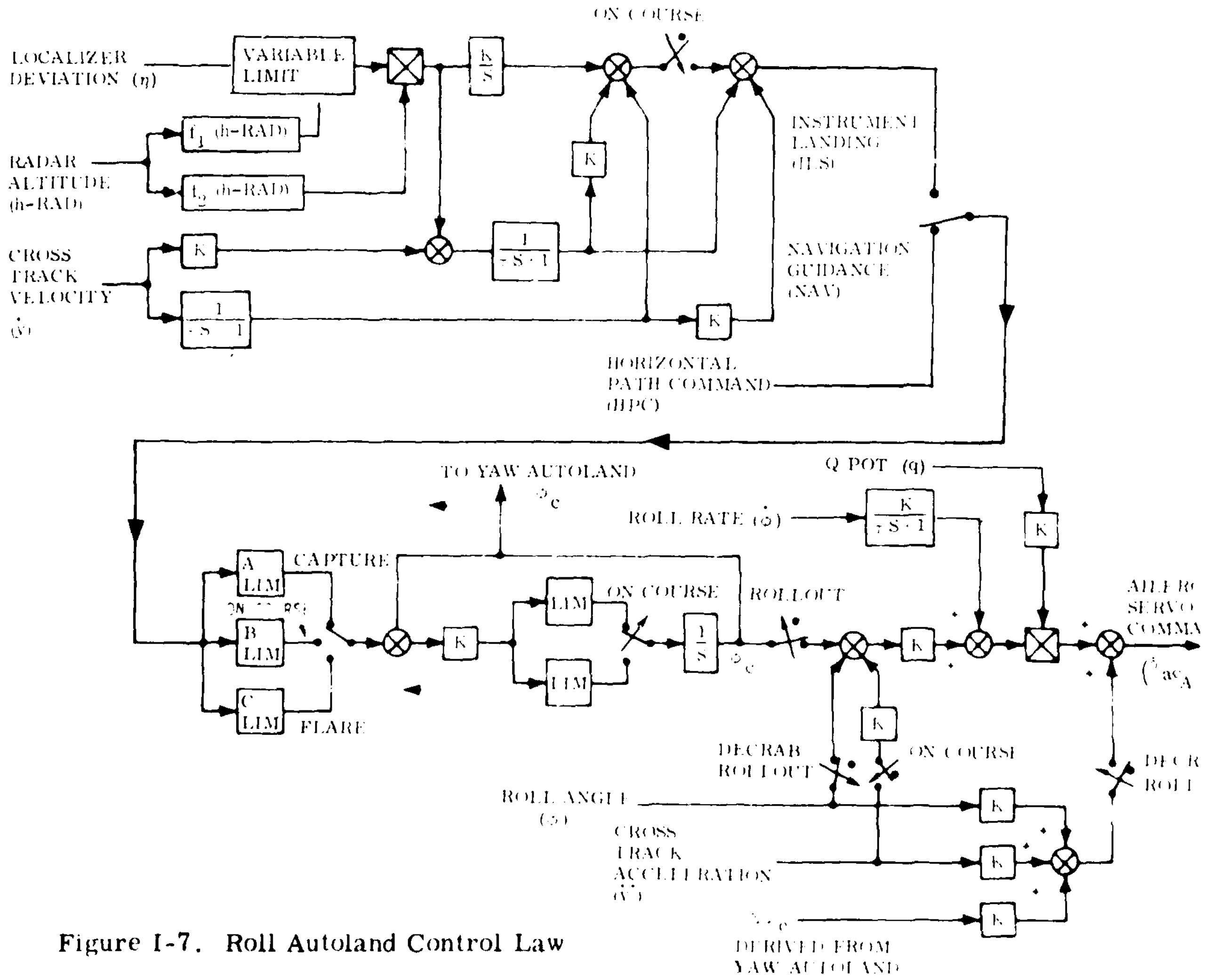


Figure I-7. Roll Autoland Control Law

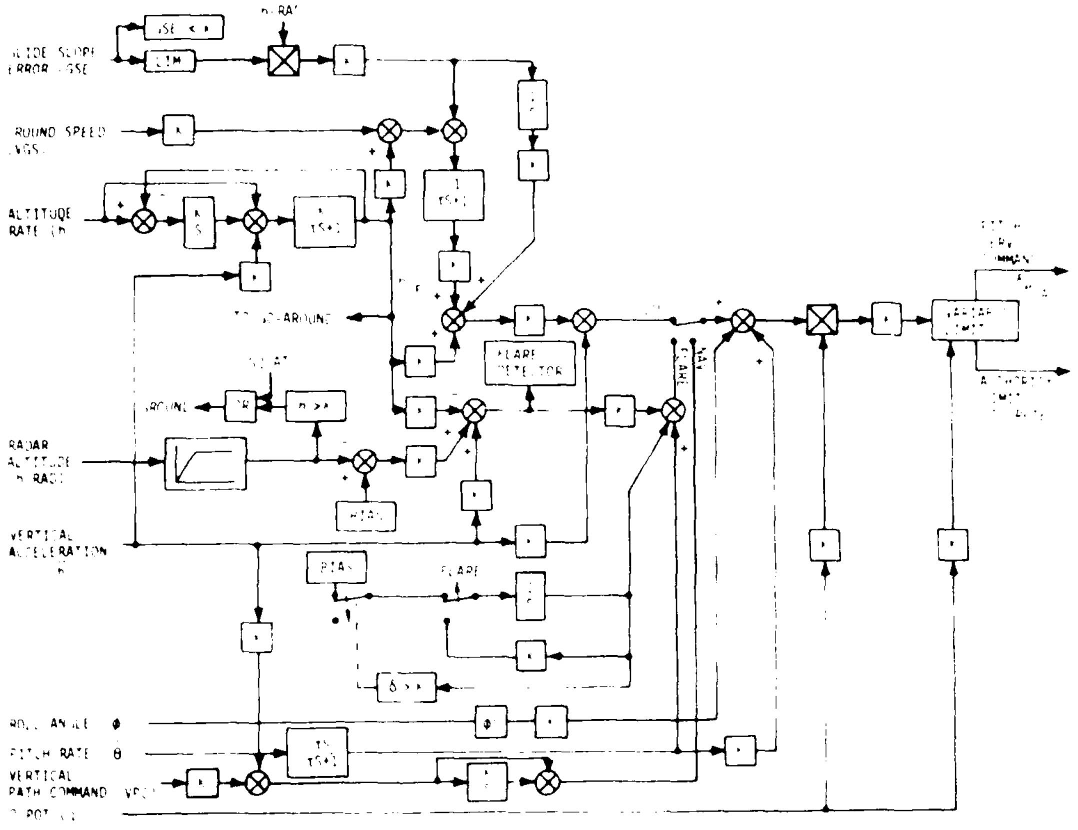


Figure I-8. Pitch Autoland Control Law

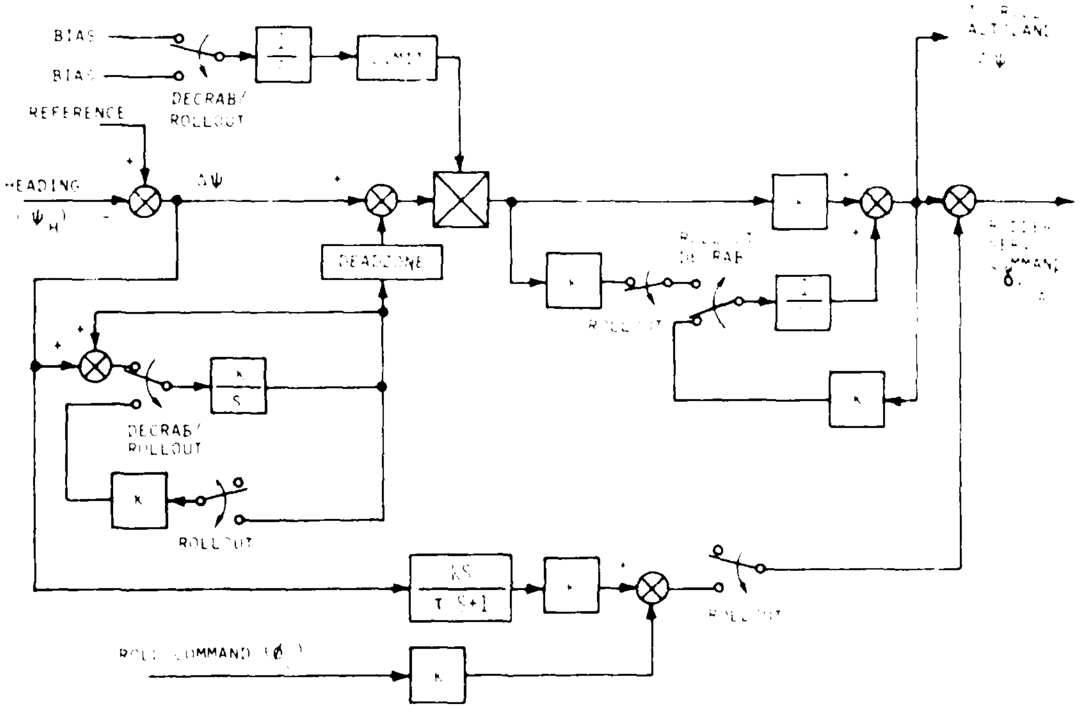


Figure I-9. Yaw Automated Control Law

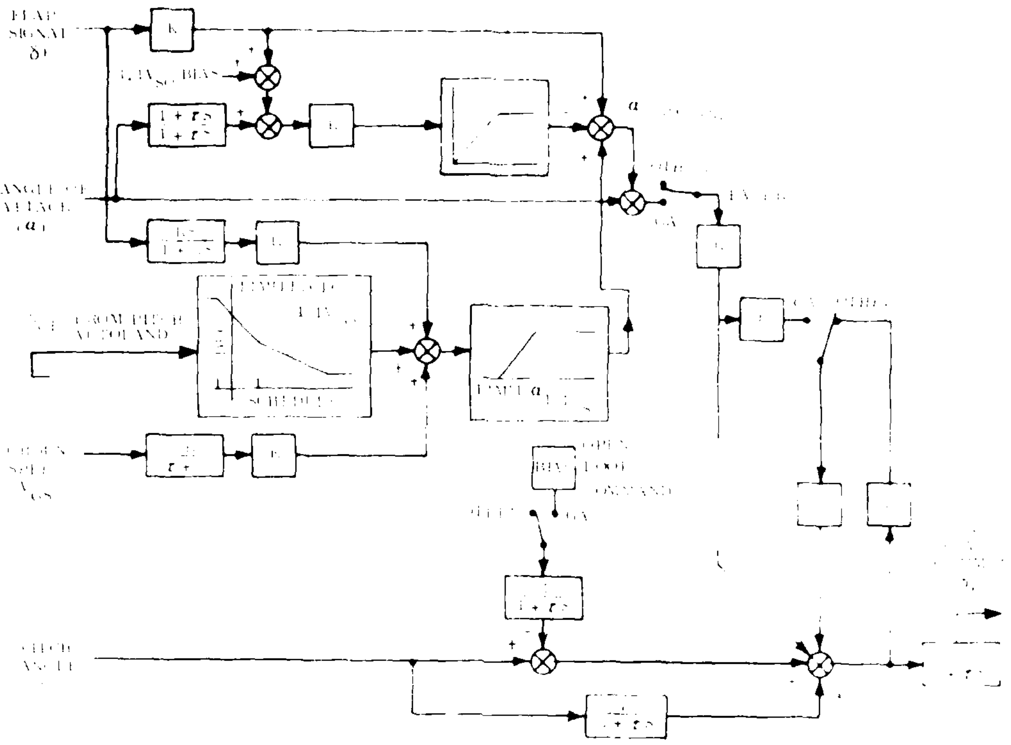


Figure I-10. Go-Around Control Law

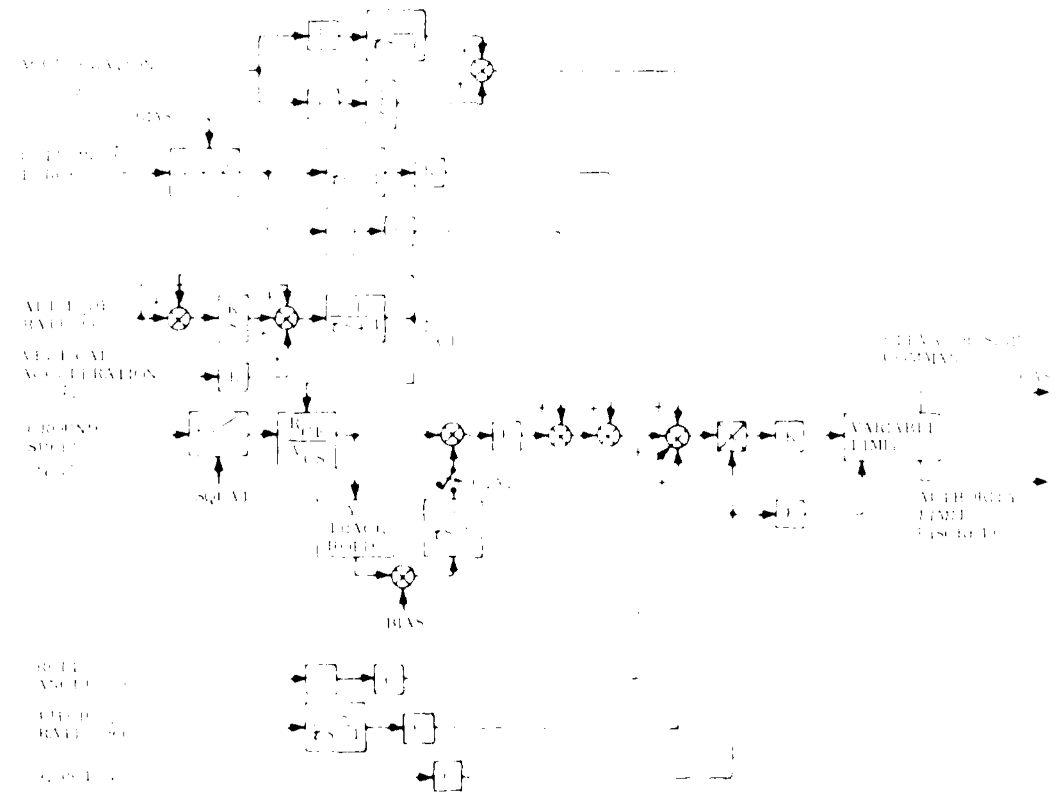


Figure I-11. Pitch Command Augmentation Control Law (with $\gamma$ Hold)

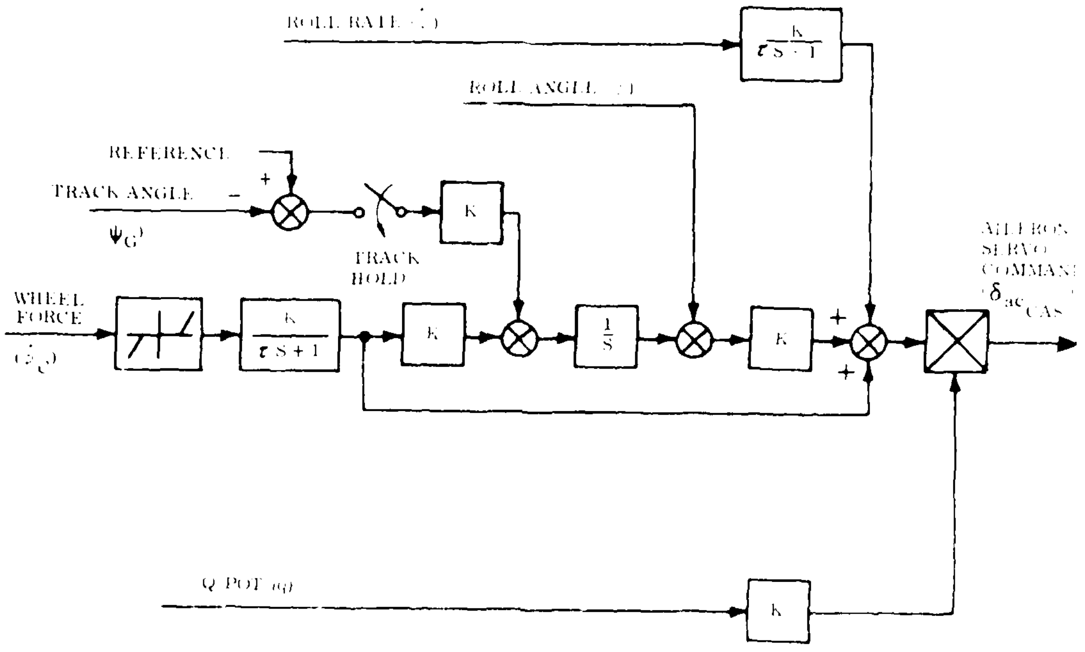


Figure I-12. Roll Command Augmentation Control Law

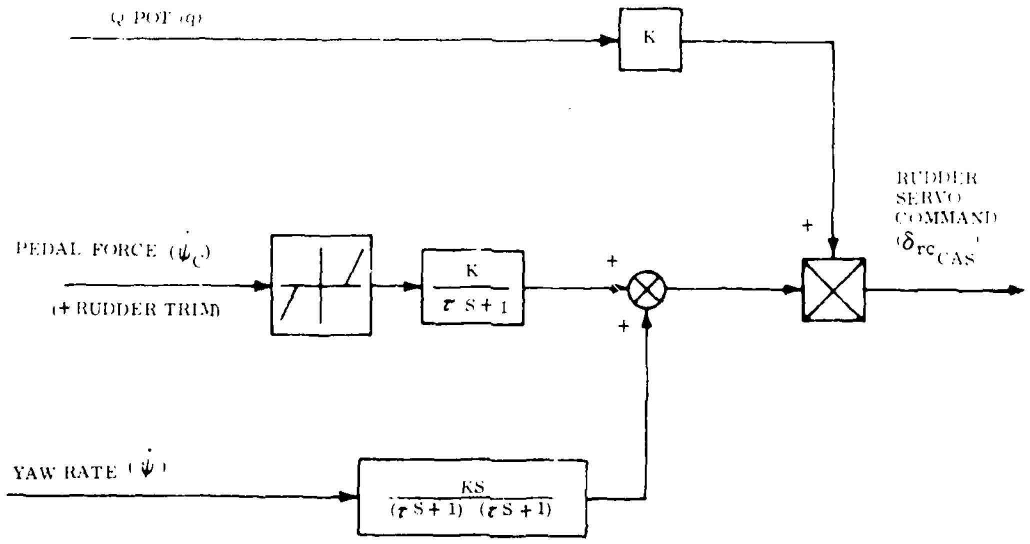


Figure I-13. Yaw Command Augmentation Control Law

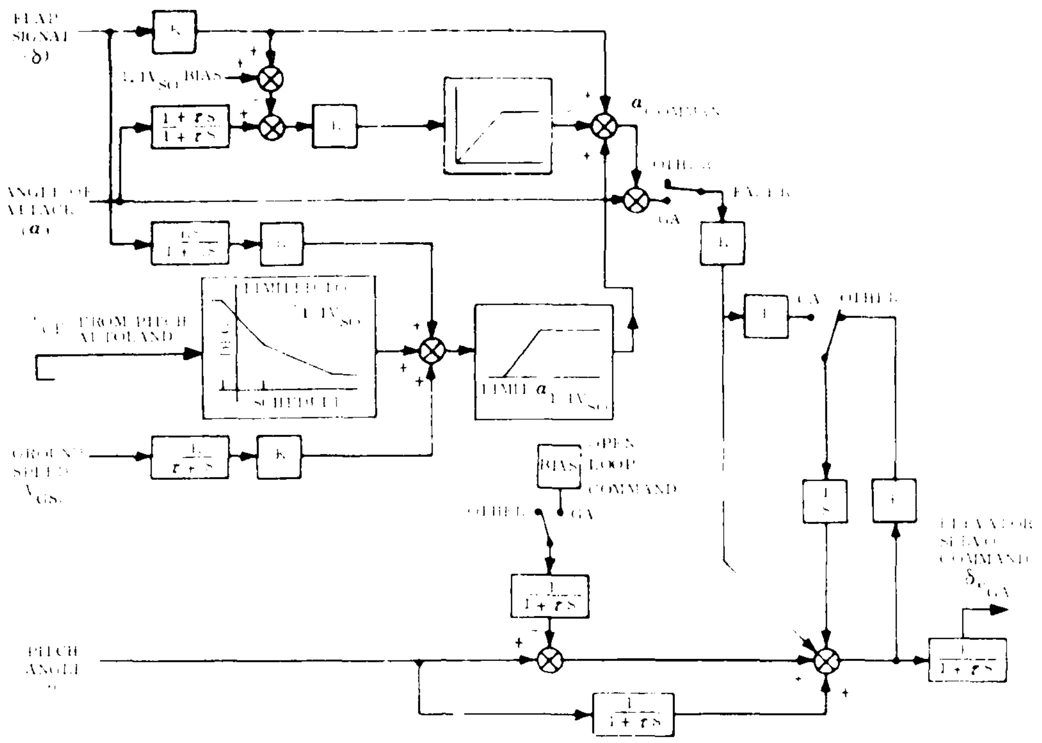


Figure I-10. Go-Around Control Law

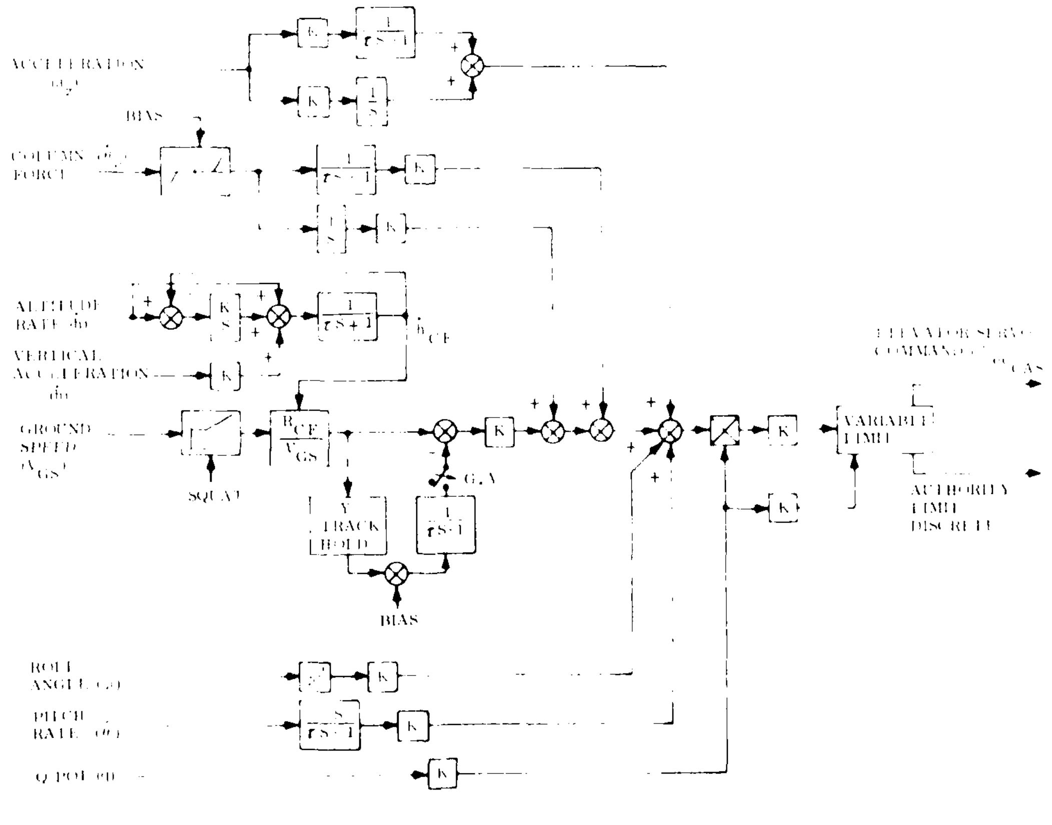


Figure I-11. Pitch Command Augmentation Control Law (with γ Hold)

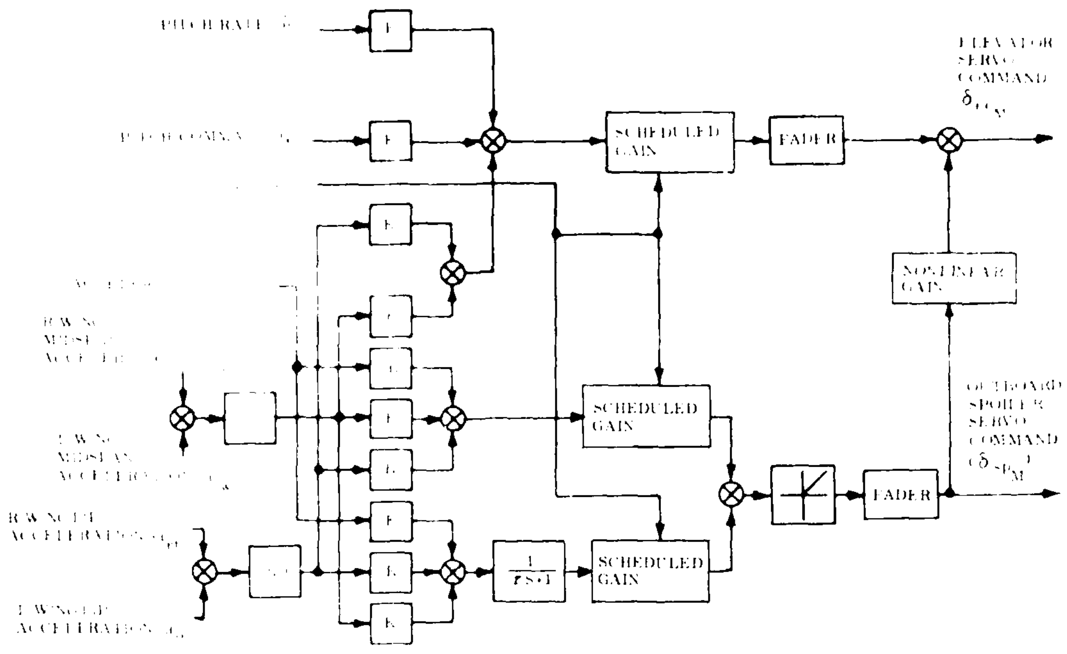
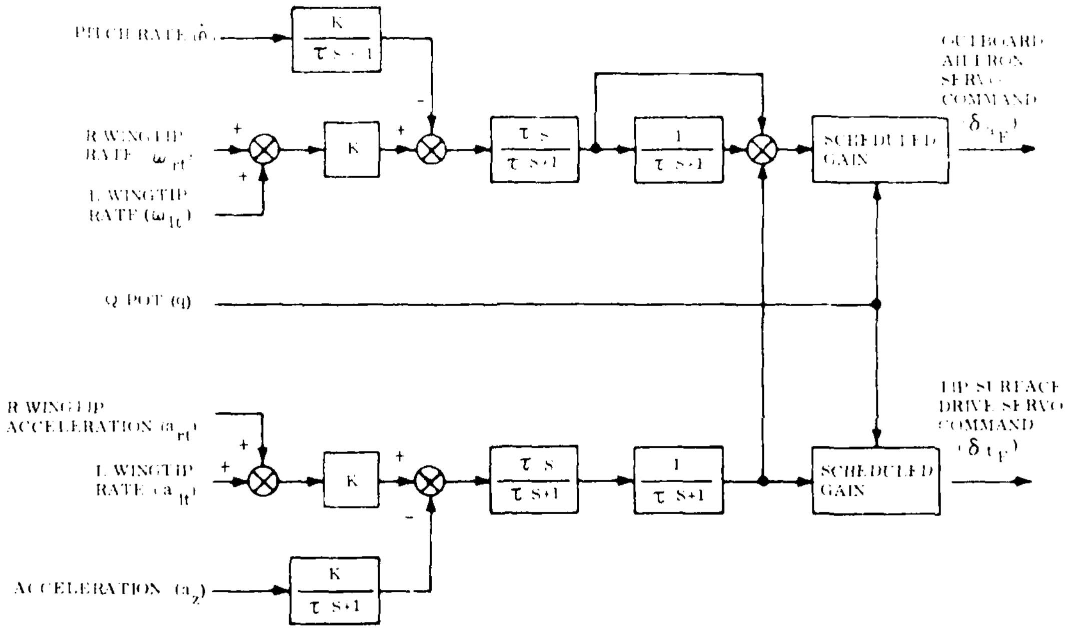


Figure I-14. Maneuver and Gust Load Alleviation Control Law



PITCH RATE
$\frac{K}{\tau s+1}$
R WINGTIP RATE $\omega_{rt}$
L WINGTIP RATE $(\omega_{lt})$
K
$\frac{\tau s}{\tau s+1}$
$\frac{1}{\tau s+1}$
SCHEDULED GAIN
OUTBOARD AILERON SERVO COMMAND $(\delta_{AF})$
Q POT (q)
R WINGTIP ACCELERATION $(a_{rt})$
L WINGTIP RATE $(a_{lt})$
ACCELERATION $(a_z)$
TIP SURFACE DRIVE SERVO COMMAND $(\delta_{tF})$

Figure I-15. Flutter Suppression Control Law

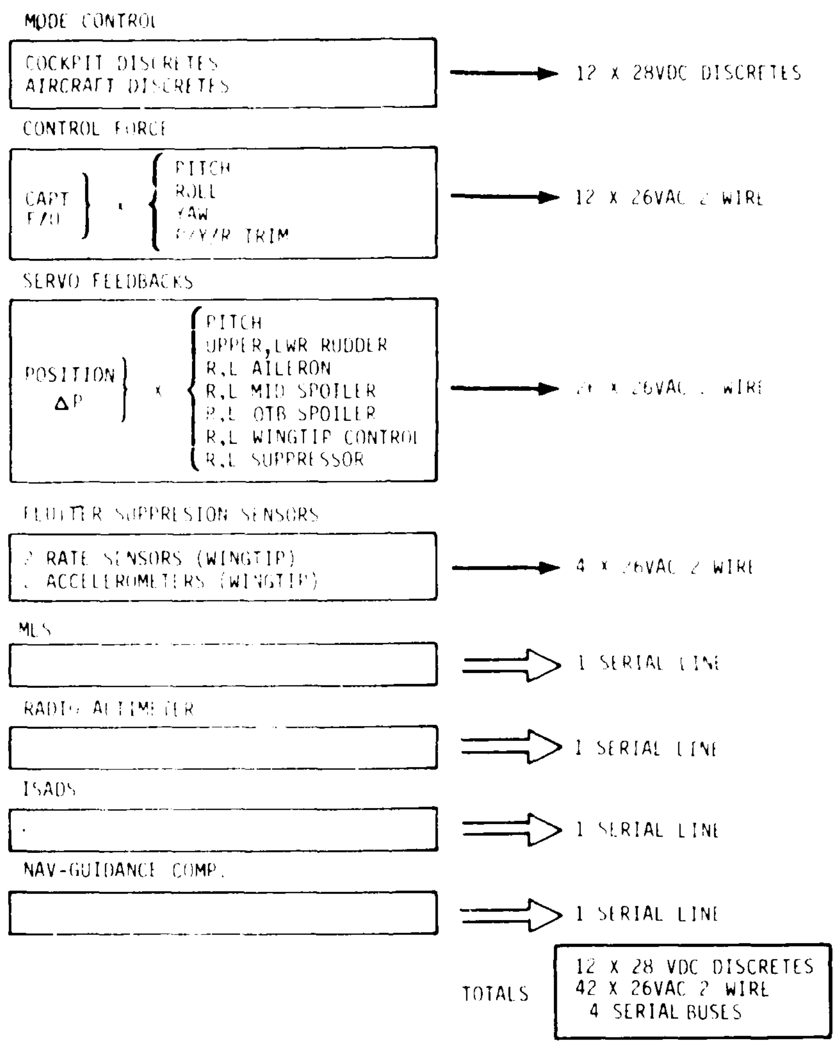


Figure I-16. Sensor and Mode Control Block

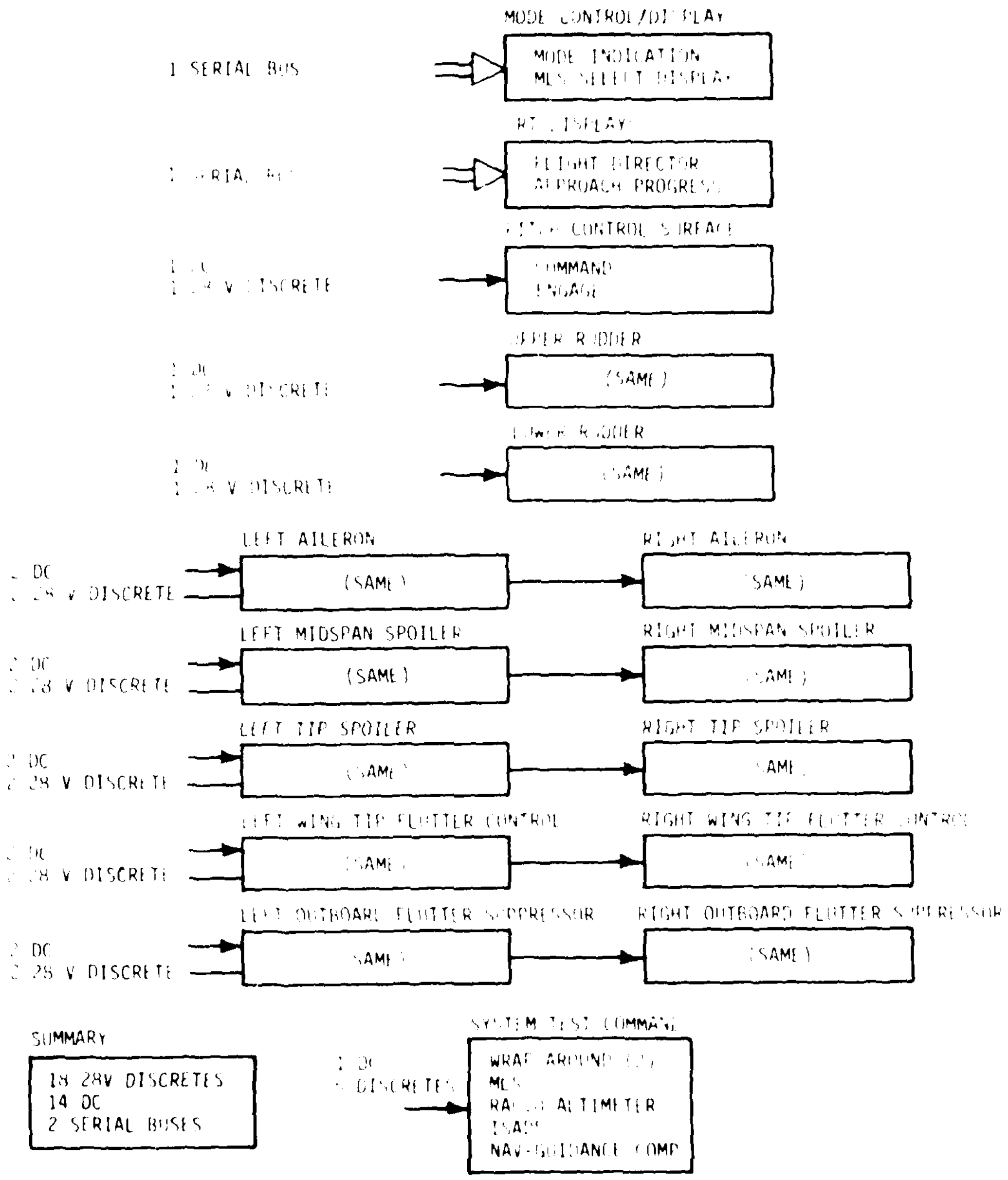


Figure I-17. Servo and Display Block

TABLE I-2. MEMORY AND TIMING ESTIMATES

| Function | Program Memory (Words) | Data Memory (Words) | Execution Rates (KOPS) |
|---|---|---|---|
| 1. Executive | 1400 | 190 | 24.00 |
| 2. Input Signal Management | 1500 | 300 | 103.60 |
| 3. Control Laws | 4500 | 500 | 176.00 |
| 4. Outer Loop Control Laws | 700 | 100 | 8.00 |
| 5. Actuator Signal | 1000 | 50 | 4.00 |
| 6. Built-in Test | 3000 | 100 | * |
| 7. Data Management | 800 | 50 | 4.00 |
| Total | 12900 | 1290 | 320.40 |

* Not applicable - either background or offline.

In all air-to-air and air-to-ground modes except TFTA, the underlying processing principle is a doppler frequency analysis of a coherent radar spectrum. Consequently, a fully coherent radar has many of the features needed in a multimode radar. However, the exact processing functions and sequence of operations differ substantially in the various modes. In the air-to-air modes, the processor is primarily concerned with the rejection of the ground return spectrum to allow detection of a comparatively weak return from an airborne target. In the high-resolution air-to-ground modes, the processing task requires a high-resolution development of the ground return spectrum into its doppler components from which a map can be generated.

The radar synthetic aperture (SAR) mode has been selected for signal processor sizing. Its block diagram is shown in Figure I-18. The beginning of the algorithmic flow is the sampled and quantized video developed by a high-speed analog-to-digital converter. The converted data are presented in bursts and temporarily stored in a buffer memory as shown in Figure I-18. After the burst has been captured in the buffer, the data rate is downshifted and all of

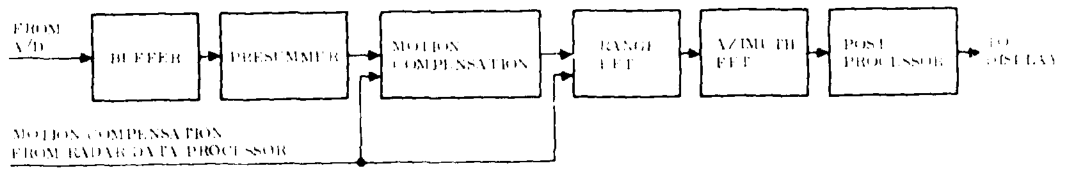


Figure I-18. Synthetic Aperture Radar Signal Processing

the remainder of the processing for this burst is accomplished in the remaining pulse repetition rate interval. This additional processing includes presumming and motion compensation using data supplied by the radar data processor. This is followed by two-dimensional transformation of the data and, finally, by post-processing. Because of the variety of missions and tasks that can be anticipated, and because the signal processor is part of a multimode radar processing string, the signal processor must be programmable.

The following implementation assessment is based on the low cost, real time processor for SAR* systems having a 5 KHz PRF and producing a 512 × 512 point map. The more detailed consideration of the operation of individual blocks of the signal processor begins with the input buffer.

## Buffer

The function of the buffer memory is to downshift the high-speed input data to the lower speed of the rest of the radar signal processor. The serial delay line buffer receives 80 MHz complex samples of two separate antenna polarizations. Each of these samples is quantized to one bit in both the I and Q channels. A total of 2048 pairs of complex samples is serially stored in four separate delay lines, one for each pulse burst. Subsequently, this stored information is serially shifted out to the presummer at a 12 MHz rate.

## Presummer

The function of this unit is to select the range samples nearest to the desired range cells and weight them in proportion to their closeness to the corresponding azimuth cell before summing them. The presummer processing sequence, shown in Figure I-19, initially stores the incoming data in a set of latches. The next step is to multiply the data by a stored reference value and add a previous value based on attitude information supplied by the radar data processor. Data thinning and compression are achieved by ignoring undesired sample data inputs and by reducing the output to 4-bit complex words consisting of two bits of I and two bits of Q channel data. As a consequence, the output data rate has been reduced to 1 MHz and, the data handling shifts from serial word processing to block processing of data arrays. Thus, storage can be centralized to a bulk working store rather than being distributed in a number of memories located in the individual processing functions.

---

* AFAL Phase I Report on Contract F33615-C-1167

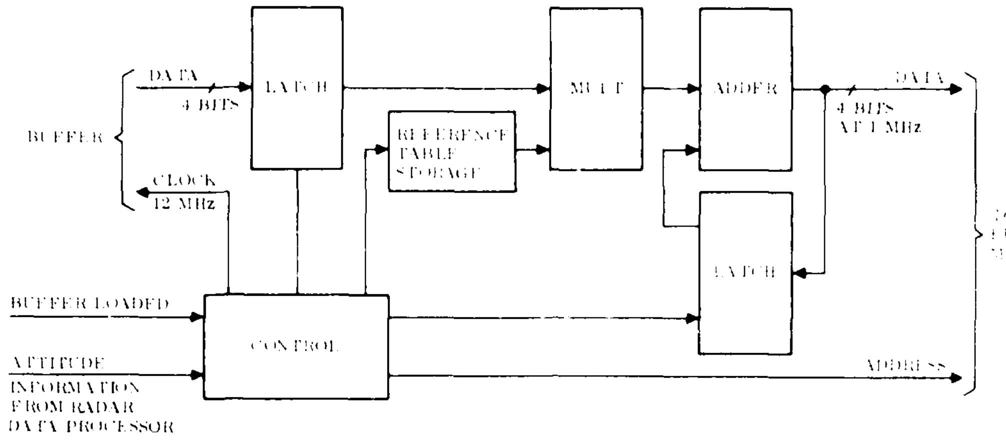


Figure I-19. Presummer

## Vector Processor

Compensation for aircraft motion is achieved through multiplication of all of the points by a two-dimensional array. This array is obtained from the radar data processor, which uses aircraft attitude system information to generate the array values. Between one million to a million and half complex multiplies are required to make these corrections.

## Two-Dimensional FFT

The two-dimensional transformation of the radar map is done in two steps. First, the ground map is transformed in the range direction by transforming 512 points from each of the 512 azimuth lines. After the range transformation has been completed, 512 512-point transforms in the azimuth direction are executed. These transformations include a computational load of about $6 \cdot 10^6$ butterflies per second on 16-bit complex data, having 8 bits of I and 8 bits of Q. Intermediate storage requirements led to the addition of a 512-word memory capable of storing complex data in addition to the use of the bulk working store.

## Post Processor

The last major functional unit in the algorithmic flow determines the magnitude of the complex data transform outputs and integrates the resulting array. The $512 \times 512$ point maps require about a million and a half operations per second on data ranging up to 16 bits.

The total SAR ground map signal processing requirements are summarized in Table I-3. These results indicate that a high throughput processor, which achieve execution rates in excess of 20 MIPS, is required. Although a large bulk memory can be employed, there is a'so a requirement for a number of smaller distributed RAM memories. Most of the storage is operated in the random access mode. The input buffer is likely to be implemented most economically in delay-line form. These estimates are based on a processor capable of performing the macro instructions listed in Addendum C.

TABLE I-3. SAR PROCESSING RATE AND STORAGE REQUIREMENTS

| Function Name | Processing Rate | Storage (Bits) |
|---|---|---|
| PRF Buffer and Presummer | $6 \times 10^6$ complex multiplies | 32.0K |
| Vector Processor (Motion Compensation) | $1.5 \times 10^6$ complex multiplies | 4.0K |
| FFT (2) | $12 \times 10^6$ butterflies sec | 8.0K |
| Bulk Memory | $2 \times 10^6$ transfers sec | 2.5M |
| Post Processor | $1.5 \times 10^6$ operations sec | 64K |
| Total | $20 \times 10^6$ complex multiplies + $2 \times 10^6$ memory accesses sec + 1.5 MIPS | 3.5M |

## ADDENDUM A

## CONTROL SURFACE COMPUTER PROCESSING FUNCTIONS AND SENSOR-ACTUATOR DIAGRAMS

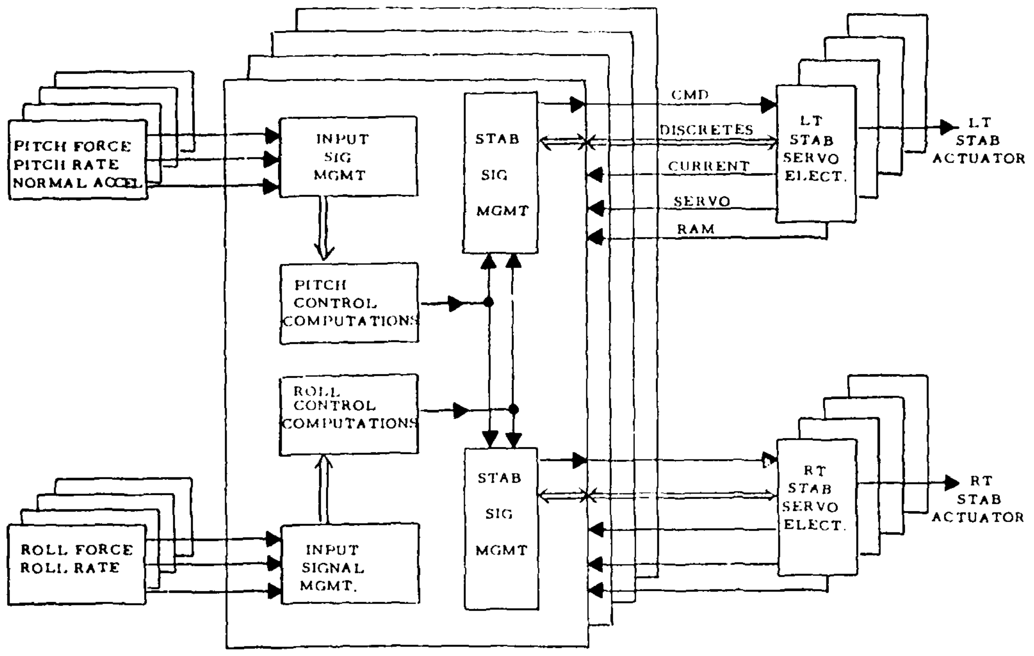


Figure A-1. Stabilator Functions

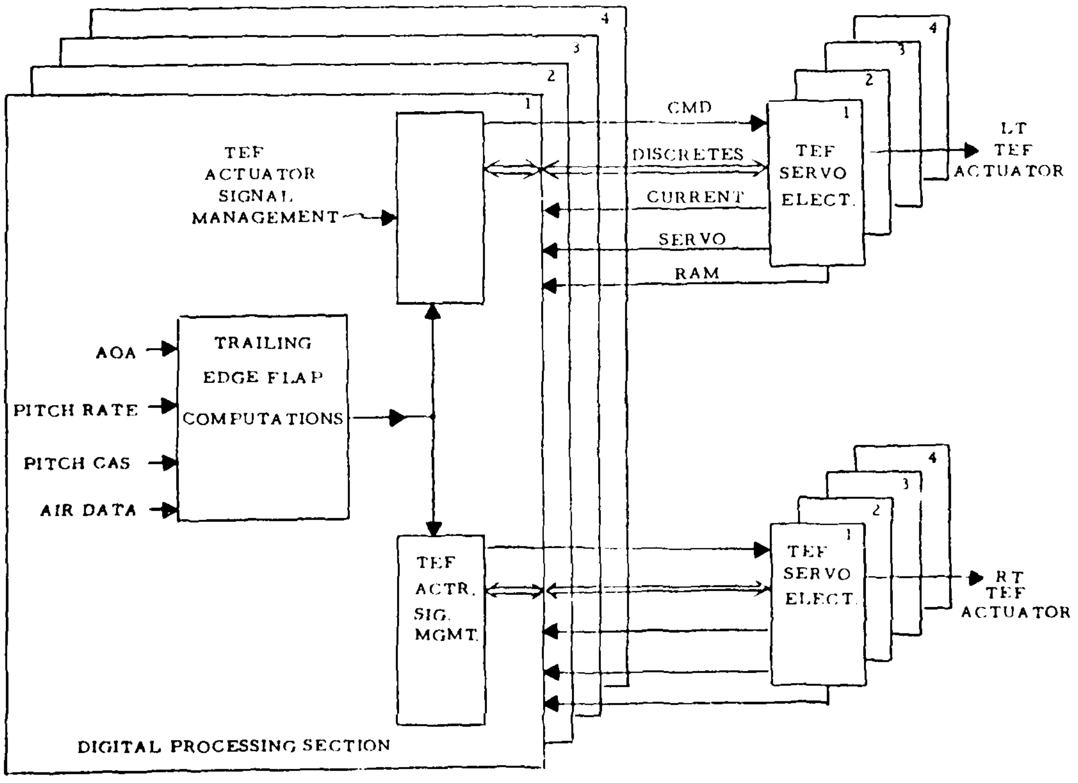


Figure A-2. Trailing Edge Flap (TEF) Functions

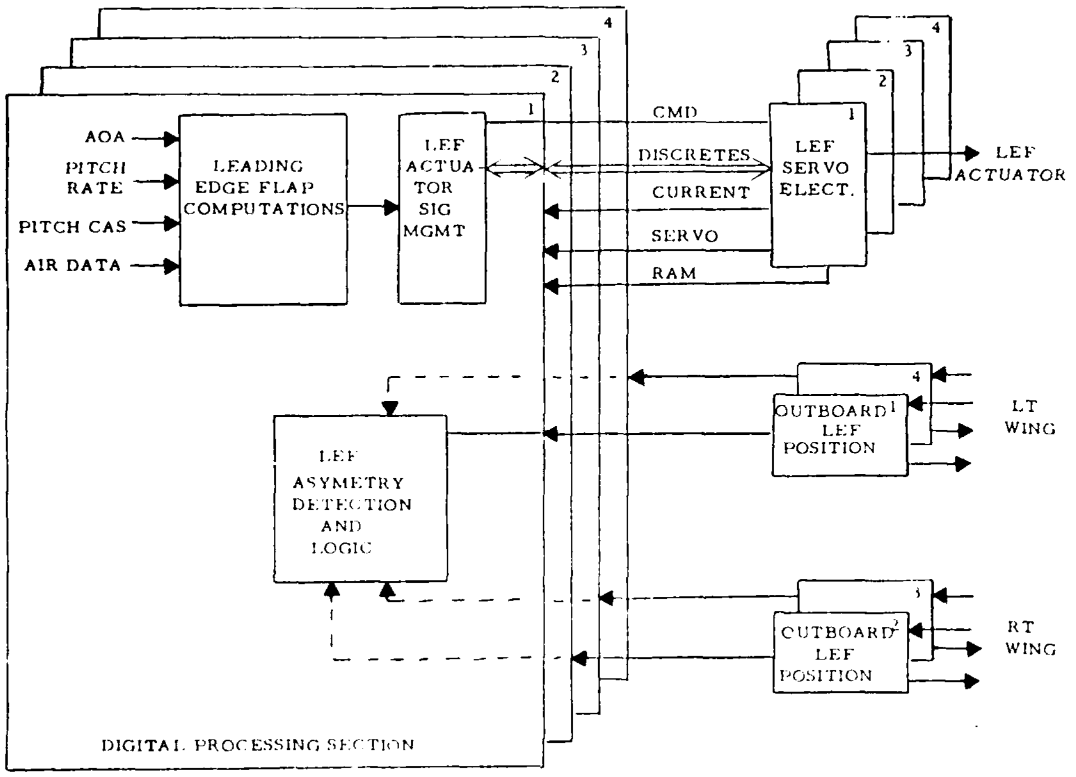


Figure A-3. Leading Edge Flap (LEF) Functions

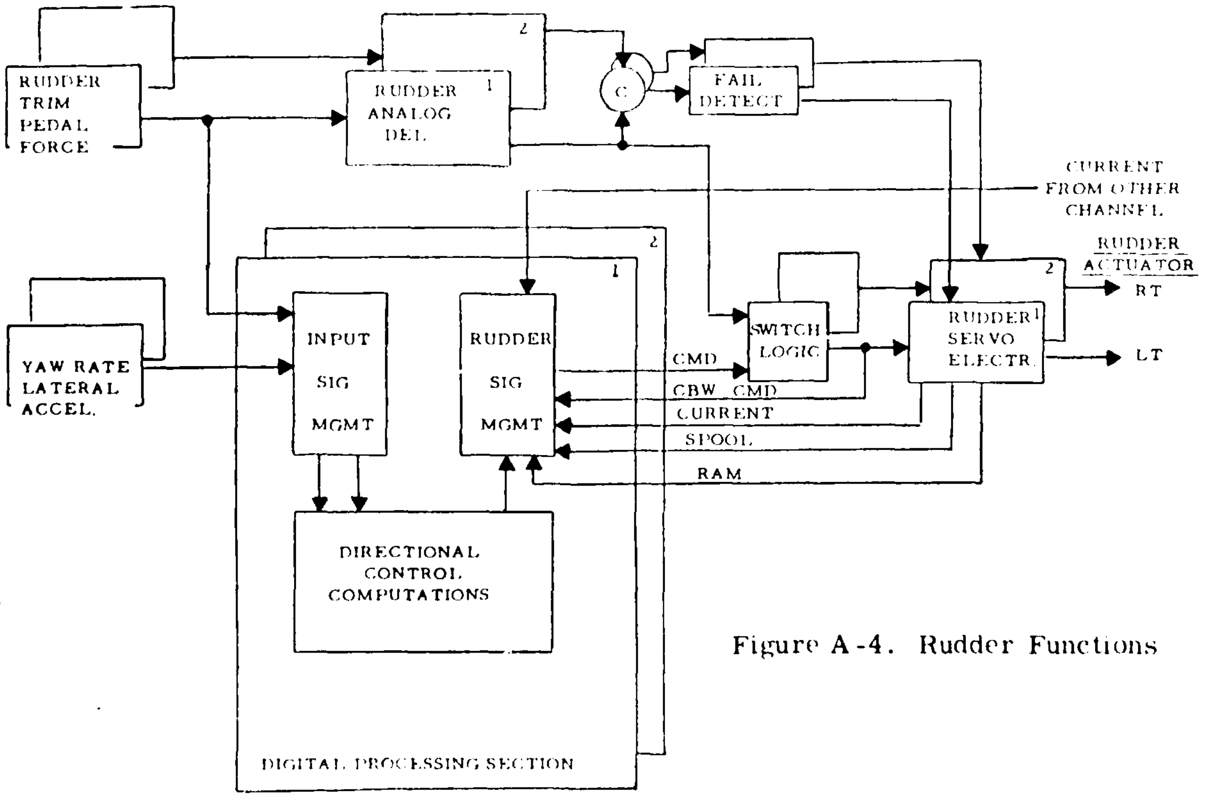


Figure A-4. Rudder Functions

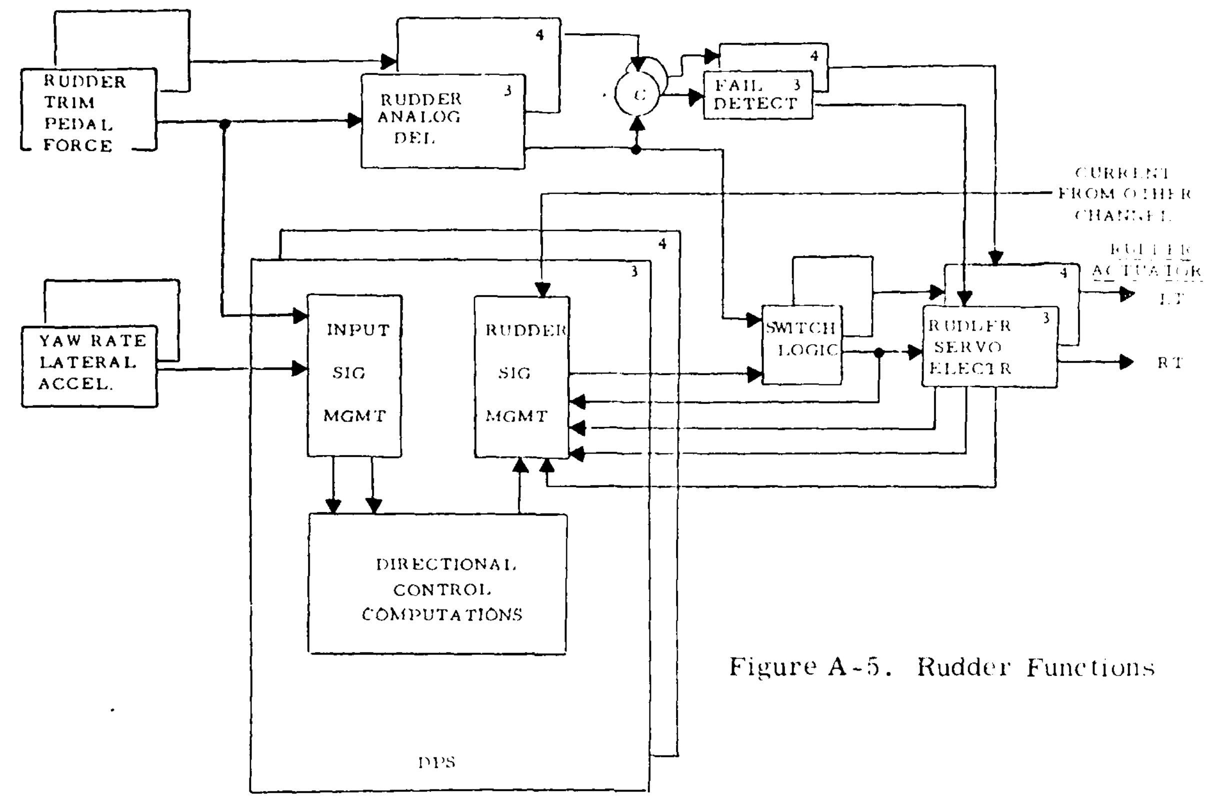


Figure A-5. Rudder Functions

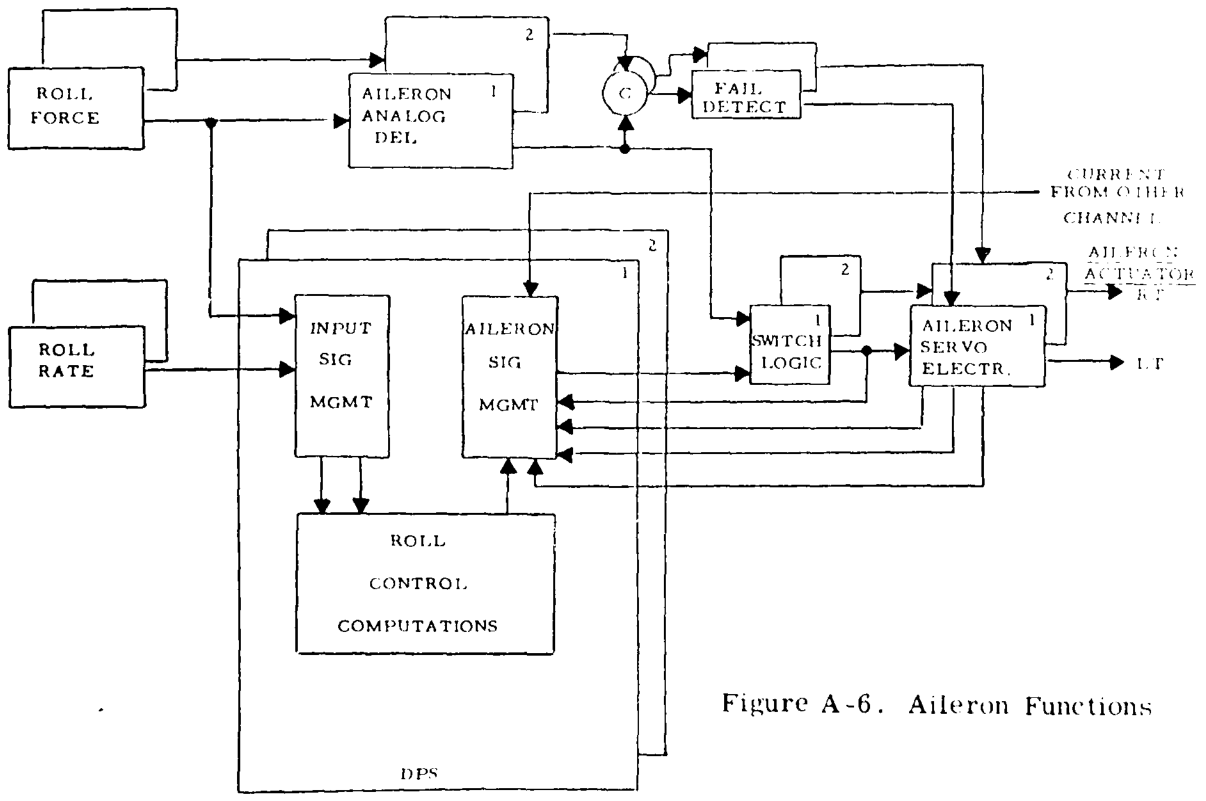


Figure A-6. Aileron Functions

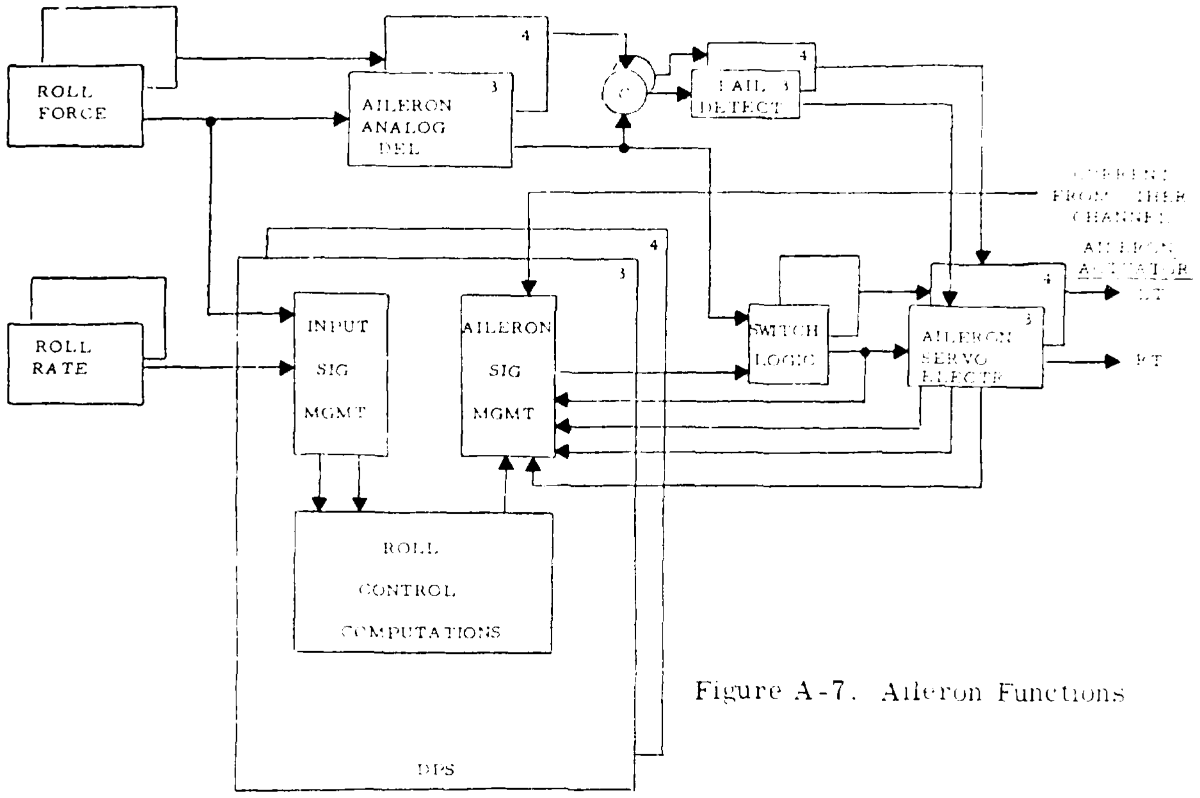


Figure A-7. Aileron Functions

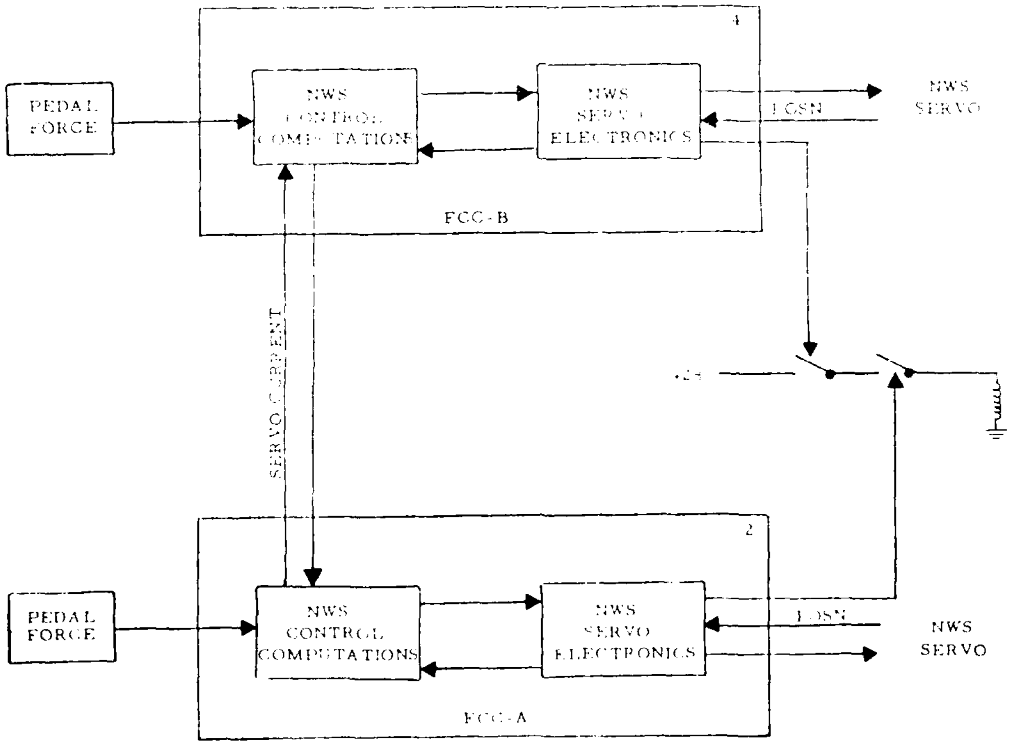


Figure A-8. Nose Wheel Steering Functions

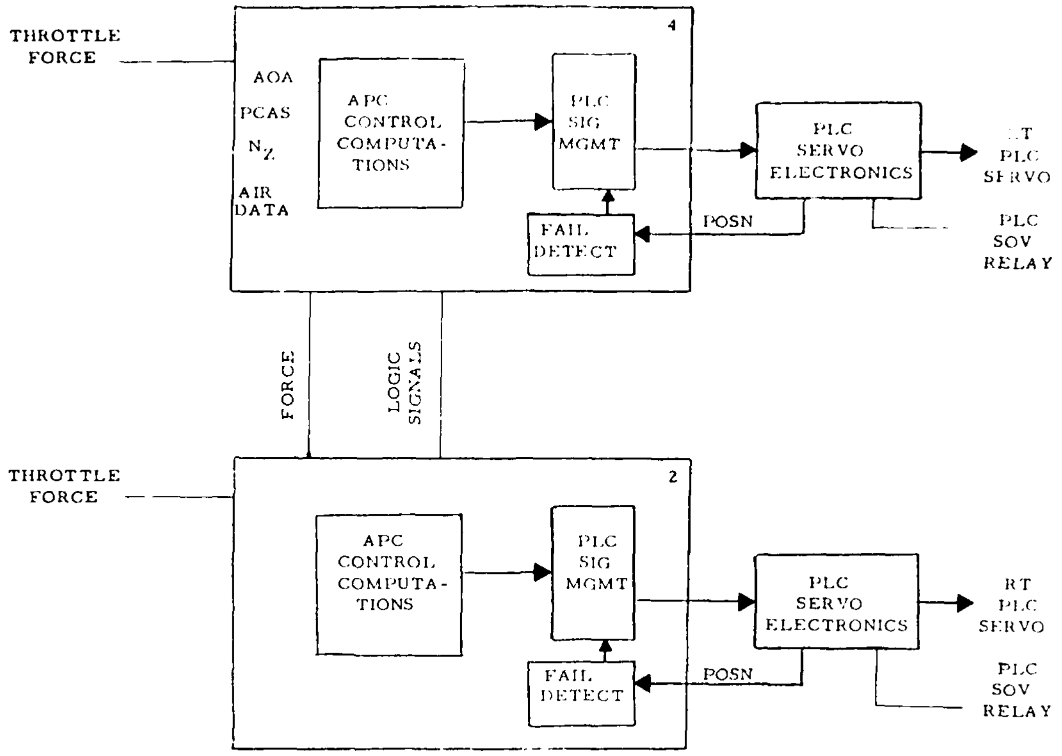


Figure A-9. Approach Power Control Functions

## ADDENDUM B

## FUNCTIONAL LISTING OF INSTRUCTIONS

## LOAD STORE INSTRUCTIONS

| Mnemonic | Instruction Description | Execution Time |
|---|---|---|
| LDU | Load UR from Program Memory | 2.0 |
| LDUS | Load UR from Scratchpad Memory | 1.5 |
| LULB | Load UR (Left Byte) Immediate | 1.25 |
| LURB | Load UR (Right Byte) Immediate | 1.25 |
| LDL | Load LR from Program Memory | 2.0 |
| LDLS | Load LR from Scratchpad Memory | 1.5 |
| LLLB | Load LR (Left Byte) Immediate | 1.25 |
| LLRB | Load LR (Right Byte) Immediate | 1.25 |
| LDA | Load XA from Program Memory | 2.0 |
| LDAS | Load XA from Scratchpad Memory | 1.5 |
| LALB | Load XA (Left Byte) Immediate | 1.25 |
| LARB | Load XA (Right Byte) Immediate | 1.25 |
| LDB | Load XB from Program Memory | 2.0 |
| LDBS | Load XB from Scratchpad Memory | 1.5 |
| LBLB | Load XB (Left Byte) Immediate | 1.25 |
| LBRB | Load XB (Right Byte) Immediate | 1.25 |
| LDC | Load XC from Program Memory | 2.0 |
| LDCS | Load XC from Scratchpad Memory | 1.5 |
| LCLB | Load XC (Left Byte) Immediate | 1.25 |
| LCRB | Load XC (Right Byte) Immediate | 1.25 |
| STU | Store UR into Program Memory | 2.0 |
| STUS | Store UR into Scratchpad Memory | 1.75 |
| STLS | Store LR into Scratchpad Memory | 1.75 |
| STAS | Store XA into Scratchpad Memory | 1.75 |
| STBS | Store XB into Scratchpad Memory | 1.75 |
| STCS | Store SC into Scratchpad Memory | 1.75 |

## ARITHMETIC INSTRUCTIONS

| Mnemonic | Instruction Description | Execution Time |
|---|---|---|
| ADU | Add to UR from Program Memory | 2.0 |
| ADUS | Add to UR from Scratchpad Memory | 1.5 |
| ADBU | Add to UR (Right Byte) Immediate | 1.25 |
| ADBL | Add to LR (Right Byte) Immediate | 1.25 |
| ADBA | Add to XA (Right Byte) Immediate | 1.25 |
| ADBB | Add to XB (Right Byte) Immediate | 1.25 |
| ADBC | Add to XC (Right Byte) Immediate | 1.25 |
| AMS | Add to Scratchpad Memory from UR | 2.5 |
| DIV | Divide UR & LR by Program Memory | 10.75 |
| DIVS | Divide UR & LR by Scratchpad Memory | 10.5 |
| MPY | Multiply UR by Program Memory | 6.0 |
| MPYS | Multiply UR by Scratchpad Memory | 5.75 |
| SBU | Subtract Scratchpad Memory from UR | 2.0 |
| SBUS | Subtract Scratchpad Memory from UR | 1.5 |

## REGISTER INSTRUCTIONS

| Mnemonic | Instruction Description | Execution Time |
|---|---|---|
| ABSU | Absolute Value of UR | 1.25 - 1.75 |
| CILB | Clear Indicator (Left Byte) Immediate | 1.25 |
| CIRB | Clear Indicator (Right Byte) Immediate | 1.25 |
| CPLU | Complement UR | 1.5 |
| INV | Invert UR | 1.25 |
| SILB | Set Indicator (Left Byte) Immediate | 1.25 |
| SIRB | Set Indicator (Right Byte) Immediate | 1.25 |
| TSU | Transfer SR to UR | 1.25 |
| TUS | Transfer UR to SR | 1.25 |
| XUA | Exchange UR and XA | 1.75 |
| XUB | Exchange UR and XB | 1.75 |
| XUC | Exchange UR and XC | 1.75 |
| XUL | Exchange UR and LR | 1.75 |

## INPUT/OUTPUT INSTRUCTIONS

| Mnemonic | Instruction Description | Execution Time |
|---|---|---|
| CLR | Clear Device Controller | 1.25 |
| CLRI | Clear Interrupt Specified | 1.25 |
| ENBL | Enable Interrupts from Device | 1.25 |
| INHB | Inhibit Interrupts from Device | 1.25 |
| SLZ | Shift UR Left - Enter Zeros | 1.25 + .25(n) |
| SLZD | Shift Double Left - Enter Zeros | 1.25 + .25(n) |
| SLZX | Shift Double Left by XC - Enter Zero | 1.25 + .25(n) |
| SRC | Shift UR Right - Circulate Bits | 1.25 + .25(n) |
| SRCD | Shift Double Right - Circulate Bits | 1.25 + .25(n) |
| SRS | Shift UR Right - Repeat Sign | 1.25 + .25(n) |
| SRSD | Shift Double Right - Repeat Sign | 1.25 + .25(n) |
| SRSX | Shift Double Right by XC - Repeat Sign | 1.25 + .25(n) |
| SRZ | Shift UR Right - Enter Zeros | 1.25 + .25(n) |
| SRZD | Shift Double Right - Enter Zeros | 1.25 + .25(n) |

## DOUBLE PRECISION INSTRUCTIONS

| Mnemonic | Instruction Description | Execution Time |
|---|---|---|
| ADD | Add Double from Program Memory | 3.0 |
| ADDS | Add Double from Scratchpad Memory | 2.5 |
| ADMS | Add Double to Scratchpad Memory | 4.25 |
| LDD | Load Double from Program Memory | 3.0 |
| LDDS | Load Double from Scratchpad Memory | 2.5 |
| STDS | Store Double into Scratchpad Memory | 3.0 |
| SBD | Subtract Double from Program Memory | 3.0 |
| SBDS | Subtract Double from Strachpad Memory | 2.5 |
| ABSD | Absolute Value of Double Register | 1.25 - 2.25 |
| CPLD | Complement Double Register | 1.75 - 2.0 |
| ZRD | Zero Dobule Register | 1.5 |
| NRM | Normalize Double Register | 2.0 + .25(n) |

## LOGICAL INSTRUCTIONS

| Mnemonic | Instruction Description | Execution Time |
|---|---|---|
| NDU | And to UR from Program Memory | 2.0 |
| NDUS | And to UR from Scratchpad Memory | 1.5 |
| ORU | Or to UR from Program Memory | 2.0 |
| ORUS | Or to UR from Scratchpad Memory | 1.5 |
| CBSP | Clear Bits Specified by Bit Mask | 2.75 |
| SBSP | Set Bits Specified by Bit Mask | 2.75 |
| SKSP | Skip on Bits Specified by Bit Mask | 2.0 - 2.25 |

## BRANCHING INSTRUCTIONS

| Mnemonic | Instruction Description | Execution Time |
|---|---|---|
| DSSZ | Decrement and Skip if Scratchpad is Zero | 2.5 - 2.75 |
| JINT | Jump to Service Interrupt | 8.0 |
| JMP | Jump Unconditional | 1.5 |
| JMPI | Jump Unconditional, Indirect | 2.0 |
| JMS | Jump to Subroutine | 1.5 |
| JMSI | Jump to Subroutine, Indirect | 2.0 |
| JSNS | Jump After Device Sense | 2.75 |
| RTN | Return from Subroutine | 1.0 |
| RINT | Return from Interrupt Routine | 5.0 |
| SIE | Skip if Program Memory Equal to UR | 2.0 - 2.2 |
| SISE | Skip if Scratchpad Memory Equal to UR | 1.75 - 2.0 |
| SIG | Skip if Program Memory Greater than UR | 2.0 - 2.2 |
| SISG | Skip if Scratchpad Memory Greater than UR | 1.75 - 2.0 |
| SIL | Skip if Program Memory Less than UR | 2.0 - 2.2 |
| SISL | Skip if Scratchpad Memory Less than UR | 1.75 - 2.0 |
| SKLB | Skip on Indicator (Left Byte) Immediate | 1.25 - 1.5 |
| SKRB | Skip on Indicator (Right Byte) Immediate | 1.75 - 2.0 |
| SKR | Skip if Device is Ready | 1.75 - 2.0 |

## ADDENDUM C

## SIGNAL PROCESSOR MACRO LISTING

## REAL VECTOR OPERATIONS

Vector Clear

Vector Move

Vector Negate

Vector Add

Vector Subtract

Vector Multiply

Vector Divide

Vector - Scaler Add

Vector - Scaler Multiply

Vector - Signed Squared

Vector Absolute Value

Vector Square Root

Vector Logaritm (Base 10)

Vector Natural Logarithm

Vector Exponential

Vector Sine

Vector Cosine

Vector Arctangent

Vector Arctangent of (Y X)

Sum of Vector Elements

Sum of Vector Squares

Dot Product of Two Vectors

Vector Float

Vector Scan and Scale (Fix)

## VECTOR MAXIMUM MINIMUM OPERATIONS

Maximum Element in a Vector

Minimum Element in a Vector

Maximum Magnitude Element in a Vector

Minimum Magnitude Element in a Vector

Maximum and Minimum of a Vector

Maximum and Minimum Magnitude of a Vector

Vector Maximum (of Two Vectors)

Vector Minimum (of Two Vectors)

Vector Maximum Magnitude of Two Vectors

Vector Minimum Magnitude of Two Vectors

## VECTOR FILTER OPERATIONS

Vector Polynomial Evaluate

Difference Equations

4 Pole Filter (Difference Equation)

## COMPLEX VECTOR OPERATIONS

Complex Vector Multiply

Complex Vector Reciprocal

Complex Vector Magnitude (Square)

Rectangular to Polar Conversion

Polar to Rectangular Conversion

## MATRIX OPERATIONS

Matrix Transpose

Matrix Multiply

Matrix Multiply (Dimension 32)

Matrix Inverse

Matrix Vector Multiply (3 · 3)

Matrix Vector Multiply (4 · 4)

## FAST FOURIER TRANSFORM OPERATIONS

Complex FFT

Real FFT

Scrambled to True Order FFT Passes

Bit-Reverse Order an Array

Real Transform Unravel Pass

## SIGNAL PROCESSING OPERATIONS

Convolution (or Correlation)

Wiener-Levinson Algorithm

Bandpass Filter

Power Spectrum

Complex Cepstrum

Inverse Complex Cepstrum

Schaffer's Phase Unwrapping

## APPENDIX II

## DESIGN GUIDELINES

# 1. INTRODUCTION

The attainment of a reliable self-diagnosing design necessitates that faults be detected when they occur. Subsequently, the errors produced by the faults must be masked or the faulty unit should be removed from the signal chain and replaced with an operational equivalent.

A knowledge of the error characteristics is necessary in order to detect these errors. This study is concerned with the errors produced by integrated semiconductor circuits, particularly large scale integrated (LSI) circuit devices utilized in processors, which includes memory and microprocessor devices. These LSI device error characteristics differ from earlier, smaller-scale devices in that one or more faults may produce one or more errors. Wang and Lovelace[1] present data that indicate that single bit errors for memory devices may represent only 75-80% of the total error population. Their work also indicates that the compositon of the failure population has a significant effect on the reliability. Consequently, error protection techniques have been required to handle both single and multiple stuck-at faults. Further attempts at characterizing the error modes have been unsuccessful, primarily because of insufficient data on available LSI devices. This is due, in part, to the recent introduction of many of the parts, but also to the relatively high obsolescent rate of some of these devices, such as random-access memory (RAM) and read-only memory (ROM) devices. The net result of this condition is that the number of errors that must be accommodated can vary between a single error to the entire set of outputs or inputs that are related, such as all of the output of a port. This model then results in the elimination of many otherwise valuable techniques.

The self-diagnosing processor must, then, be able to detect multiple internal errors and determine the location of the failure with sufficient resolution so that the subsequent maintenance action is quick and effective. This approach has the advantage of:

1) Easy error detection, since the errors are detected (usually) upon their first occurrence. The operation of the processor in the failed state is considered during the design.

2) Automatic detection of a large percentage of errors. Few undetected errors occur.

3) Simple, fast diagnosis due to built-in error detection.

4) More effective handling of inconsistent errors, such as intermittents and transients. Diagnosis is initiated immediately upon detection of the error.

1 Wang, S.O., Lovelace, K., "Improvement of Memory Reliability by Single Bit Error Correction", COMCON 77.

5) Manual maintenance is simplified, computer maintenance costs are reduced.

Fault detection techniques have been emphasized because of the poor reliability of systems possessing less than complete fault detection. For systems based on standby principles, Losq[2] has shown that the coverage affects the reliability in two ways — it reduces the maximum reliability of the system and it modifies the shape of the perfect coverage reliability curve by a factor of

$$\exp(-\lambda(1-c)T),$$

where

$\lambda$ = failure rate,

$c$ = coverage in percent, and

$T$ = time interval

The price of increased fault tolerance obtained through the commitment of additional hardware resources is an increase in failure probability due to these added resources. Techniques that provide reliability enhancement and self-diagnosis are particularly effective for these applications provided that the maintenance intervals are short compared to the system MTBF.

The resulting guidelines should

1) provide computational capability that is consistent with the previously identified baseline processing requirements,

2) provide a modular processor architecture that is adaptable to changing requirements driven by either changing mission or variety of application requirements,

3) match the 1977 technology and maintain flexibility with respect to anticipated improvements in the state of the art.

2 Losq, J., "Influence of Fault Detection and Switching Mechanisms on the Reliability of Standby Systems", FTC 75

## II. REVIEW OF RELIABILITY TECHNIQUES FOR LSI DEVICES

There are two basic techniques for self-diagnosing systems with automatic detection:

1) The information signals of the system are encoded in such a manner that the signals form a code word in an error detecting correcting code under fault-free conditions. When a detectable fault occurs, an error is produced that is a non-code word. An example of a single error-detecting code is replication with comparison voting.

2) Periodic diagnosis of all modules for error detection.

By itself, the second technique is not recommended for fault diagnosis because:

1) Inconsistent errors may not be detected and their effect on the state and data base of the system cannot be predicted.

2) Rollback restart snapshots of the state-of-the-machine requirements are frequently not consistent with real time applications.

3) Diagnostic tools for producing test vectors for multiple faults and errors are only now being developed and the fault location accuracy is quite suspect.

Combinations of these two basic approaches are also utilized and will be discussed. The periodic approach will most likely be the basis of the manual diagnosis that supplements the automatic on-line error detection and locates the failed component to the self-diagnosing processor replaceable module level. Thus, the overall maintenance action is a combination of the spatial redundancy of the coding approach and the temporal redundancy of the periodic diagnosis.

### 1. Coding Techniques

The simplest codes are those of replication combined with a form of comparison. Duplication with comparison is perhaps the simplest error detecting technique. In this technique, two independent systems compute the same function and the results are compared to detect differences. When an error exists, the results of the independent systems will differ and the comparison will detect the difference. The comparison can usually provide location information (identify the bit location) if desired.

The simplest error correcting technique using coding involves the use of triplication. In this technique, three independent systems compute the same function and the output is the majority function of the results of the three systems. This voting of the output is performed on a signal-by-signal basis and the effect of single errors is masked. Such systems are commonly designated as triple-modular redundancy (TMR). As in duplication with comparison, location information can be derived.

Extension of the degree of replication beyond three has been labeled N-modular redundancy (NMR), where N modules are used to execute the same function and $(N-1)/2$ or fewer failures are masked. The majority function, which produces the output, has N inputs and a threshold equal to the largest integer greater than or equal to $N/2$. As with other replication schemes, location can be derived at the cost of additional resources.

Variations of the replication code include systems that both correct errors and locate the source of the error. As errored modules are identified, they are switched out of the system and replaced by standby modules. When inserting the standby modules into the system, any internal memory must be initialized.

Another variation is an adaptive technique, where the threshold of the majority function is reduced as errors are detected and the offending module is switched out of the system. Initially, the threshold is set to $N/2$; then, as errors are detected, the threshold is lowered to $(N-1)/2$, $(N-2)/2$, $(N-3)/2$, and so on, until the number of modules is reduced to two or three.

## 2. Conventional Coding Techniques

Conventional coding techniques have been developed that detect and/or correct errors. Only a subset of all the coding techniques is of interest for this study. This subset is useful for checking computations and is usually restricted to binary codes or codes that are closely related to them. A successful utilization of computational coding techniques in self-diagnosing systems depends largely on the nature of the function to be protected by the code. Hence, after an initial general discussion of codes, the effectiveness of coding techniques will be considered with respect to the memory, processor, control, and internal buses of a self-diagnosing processor.

The computational codes that were considered are the linear block codes, the arithmetic codes, and checksum codes. Hamming and parity and $\bar{b}$-adjacent codes were the linear block codes specifically evaluated for transmission and storage. Arithmetic codes were examined, primarily with respect to arithmetic operations, although their use for the protection of storage was examined. So-called low cost codes, as defined by Avizienis[3], received most of the attention. These include the AN and residue codes. The b-bit byte checksum codes examined were those having a check symbol of the form $2^b$.

3 Avizienis, A., "Digital Fault Diagnosis by Low Cost Arithmetic Coding Techniques", Proc. Purdue Centennial Year Symp. Information Processing, 1:81-91.

Application of these techniques to large-scale integrated logic circuits has not generally been successful in the sense that the implementations were low cost. There are a number of reasons for this: first, for devices such as a microprocessor, there is a mixture of structures and operations that the code must span if it is to be applied external to the device. Since the codes, in general, are matched to the structure and operation, this method of attack leads to difficulties that are still unsolved.

If the coding techniques are applied within the device, the size of the chip must be expanded and the number of pins increases unless the pin can be time shared. Generally, this is not possible. Applying coding techniques within a device of the size of a microprocessor at the register-to-register level leads to a redesign of the function and usually necessitates an increase in chip size. Since these LSI devices are already at or near the current state-of-the-art in integration, the increase in chip size results in a loss of yield, which, already, is relatively low at least compared to small-scale integration (SSI). Since the coding techniques examined thus far result in an increase in circuitry of at least twice the original device, the application of coding techniques results in uneconomical designs because of the low yield.

The second major source of difficulty in applying coding techniques to highly integrated devices is the lack of good error models that relate faults and errors. As indicated in the Introduction, it is believed that single stuck-at fault modeling results in insufficient fault coverage. It has been shown that the effectiveness of the error detection is very sensitive to coverage, particularly in the range of interest for self-diagnosis. Hence, it has been decided that multiple errors must be considered. Implementation costs of multiple error codes in the range of four to eight bits has been found to rise rapidly. Even for the so-called unidirectional faults[4], the implementation costs increase significantly and the computational delays increase with increasing word length.

As will be seen in the application of redundancy techniques to the various functional units of a processor, most redundancy techniques that are theoretically interesting are only applicable at the component level. Technology constraints at the LSI level of integration tend to dictate that redundancy should be applied over chips not within the components of chips. Hence, relatively few redundancy techniques remain relevant. Consequently, architectural considerations are of primary importance in the design of a self-diagnosing processor. Recovery from the fault, beginning with the processing of any locati[illegible]formation through, possibly, the restoration of the processing and redundan[illegible] the major issue. As will be seen, however, a number of these redundancy techniques, originally intended for low-level application, form the basis for enhancing system reliability.

4 All components of the error value have the same sign. That is, the only erroneous bits are either 1's changed to 0's or 0's changed to 1's but not both.

3. Processors

Bit-slice microprocessors such as the AMD 2901, in contrast to monolithic microprocessors, allow the use of redundancy techniques at a lower level and, therefore, are better candidates for redundancy techniques. Both arithmetic and parity prediction techniques have been successfully applied to arithmetic operations and can be effectively used for the protection of the microprocessor arithmetic logic unit (ALU). But these techniques do not check logical operations. Of greater consequence is the fact that low-cost coding techniques for checking logical operations for error detection have not been discovered. It is generally accepted that the simplest error detecting codes for logical operations amount to duplication. Previous implementations of error detecting designs frequently resorted to duplication. Variants of these codes have been developed for ripple carry arithmetic units and carry look-ahead designs in which the carry circuits are disabled. For bit-slice microprocessors similar to the 2900 series, which incorporate byte carry look-ahead, the effect of incorporating these arithmetic coding techniques is significant. High-speed arithmetic structures, using cascades of 2901's and one or more special carry look-ahead devices, can be implemented. Speedup techniques such as these considerably increase the execution speed of these structures compared with ripple-carry techniques, particularly for long words. Addition of this code circuitry to the microprocessor and the high-speed carry look-ahead would reduce the execution speed of the microprocessor and, possibly, the high-speed carry look-ahead. Also, larger chips would be required to implement this additional circutiry barring the use of higher density fabrication techniques. Hence, this development was not pursued farther.

Alternatively, the logical operations can be removed from the ALU and implemented separately. But this approach suffers from increased delay penalties as well as the increased implementation costs cited. Implementation of the logic execution circuitry external to the microprocessor suffers from the difficulty that one or more of the operands must come from the microprocessor's register file through the output port. This increases the execution time for single-register file sourced operands and, probably, would double the cycle time for two register file sourced operands. Thus, this approach was also not recommended.

Instead, it was concluded that for a self-diagnosing computer, replication offered the best trade-off in terms of protection and implementation and execution time resources for the current state-of-the-art of integrated circuit technology. Conventional coding techniques were less effective for the multiple error case, when applied to the highly integrated LSI devices, than the single error case associated with the mdeium and small-scale integrated circuits.

For monolithic microprocessors, replication appears to be the best solution. As will be seen, after memory and bus functions are examined, TMR is believed to be the best general solution for monolithic processor applications.

## 4. Memory

Conventional coding techniques are much more favorably applied to memory functions than processors. Part of the reason is that coder decoders for arithmetic and arbitrary logic are just about as complex as the function they are protecting. But this is not true for memories. For instance, the encoder/decoder for a 4K by 29 bit/word memory[5] can be realized for about 7% of the simple memory resources. In addition, semiconductor LSI memory devices have developed in such a way that many of the problems associated with other memory technologies have been eliminated. Addressing errors are confined to a single chip under the single fault assumption by including on each chip its own decoder, along with the read and write amplifiers and read/write control circuitry.

The evaluation of the implementation requirements of codes for the protection of memory is primarily concerned with three major contributors:

1) The number of redundant or check bits that must be added on a per word basis.

2) The complexity of the associated encoders and decoders.

3) Additional delay incurred as a result of adding protection devices since these delays usually increase the address and/or the cycle time of the memory

Wensley, et al[5], discuss the bounds on redundancy codes, properties of Hamming, Hong Patel, Abramson, and Gilbert codes that almost achieve these bounds, and the performance of these error correcting codes. The lower bounds for the number of redundant digits, r, as a function of the number of information bits, k, are listed in Table II-1 which is taken from Wensley[5]. The bound varies with the number of burst code bits in a protected byte of width b, and the type of code. The $S_C$ columns list the number of digits required for cyclic burst binary codes and the $S_F$ columns list those for the single byte correcting codes. (Note that r in bits is equal to b multiplied by the entry in the appropriate column, either $S_C$ or $S_F$.) Hamming codes, with b = 1 and the Hong Patel codes for b = 2 and 4, achieve the redundancy implied by the entries listed in the $S_F$ columns.

An indication of the decoder complexity for a particular cellular decoding scheme for generalized Hamming codes discussed by Wensley in (5) is presented in Table II-2 for a 24 bit word memory of 4096 words. Memory chip size is maintained at 4096 by configuring the chips as follows: (1 bit wide × 4096), (2 bits wide × 2048), (4 bits wide × 1024), and 8 bits wide × 512). The codes are single, b bit wide, error correcting Hamming codes. For this implementation, the 2-bit wide byte (2 bits wide × 2048 word memory chip) yields the best design in terms of number of implementation costs, i.e., number of chips.

5 Wensley, J.H., Levitt, K.N., Green, M.W., Goldber, J, and Neumann, "Design of a Fault Tolerant Airborne Digital Computer", Vol. I, Stanford Research Institute, N74-17909, p., 26.

TABLE II-1. REDUNDANCY FOR CORRECTING CYCLIC AND BYTE BURSTS

# CHECK DIGITS r

| k BITS | b = 1 | b = 2 | | b = 4 | |
|---|---|---|---|---|---|
| | | $S_C$ | $S_F$ | $S_C$ | $S_F$ |
| 4 | 3 | 5 | 4 | 7 | 6 |
| 8 | 4 | 5 | 5 | 7 | 6 |
| 12 | 5 | 6 | 5 | 8 | 7 |
| 16 | 5 | 6 | 6 | 8 | 7 |
| 24 | 5 | 6 | 6 | 9 | 7 |
| 28 | 6 | 7 | 6 | 9 | 8 |
| 56 | 6 | 7 | 7 | 10 | 8 |
| 60 | 6 | 8 | 7 | 10 | 8 |
| 64 | 7 | 8 | 7 | 10 | 9 |
| 128 | 8 | 9 | 8 | 11 | 10 |
| 256 | 9 | 10 | 9 | 12 | 10 |
| 512 | 10 | 11 | 10 | 13 | 11 |
| 1024 | 11 | 12 | 11 | 14 | 12 |

TABLE II-2. SUMMARY OF DECODER COMPLEXITIES FOR CELLULAR DECODING OF GENERALIZED HAMMING CODES

| MEMORY CHIP CONFIGURATION | # OF "CHECKS" FRAMES | # OF FRAMES (TOTAL) | # OF CHIPS TOTAL | # OF DECODER GATES |
|---|---|---|---|---|
| 1 × 4096 | 5 | 29 | 29 | 1560 |
| 2 × 2048 | 3 | 15 | 30 | 936 |
| 4 × 1024 | 2 | 8 | 32 | 2496 |
| 8 × 512 | 2 | 5 | 40 | 19968 |

FRAME    NUMBER OF BITS PER WORD OF THE SELECTED MEMORY DEVICE

# FRAMES (TOTAL)    NUMBER OF FRAMES (DEVICES) IN ONE ROW OF MEMORY DEVICES

These codes mask all single byte errors. To achieve diagnosis of the errors, they should be augmented to provide detection. In some cases, this will change the number of redundant check bits and the decoder complexity. However, it is still believed that the optimum byte width is b = 2.

Another less elegant coding approach is to employ parity. Jack, et al[6], has compared various versions of parity with checksum and Hamming codes for the purpose of achieving a self-checking[7] semiconductor memory. In this paper, Jack et al, points out the importance of coverage, particularly with respect to multiple errors in memory devices. Three different forms of parity are considered. They are:

1) Single bit parity across the entire memory word.

2) Byte wide parity across 8-bit bytes using a single parity bit per byte.

3) Chip-wide parity provides one parity group for each bit position in a chip for a group of bytes. For a 16-bit data word implemented using 4-bit wide memory devices, four parity check bits are required with four parity checkers.

From coverage considerations, they conclude that, on the average, chip-wide parity and Hamming-like codes provide the best self-checking coverage for data faults of any of the detection approaches investigated. They are also unsurpassed in terms of worst-case possible failure modes. In presenting the results of the coverage analysis, they note that no exact overall coverage figures can be determined unless all the failure modes and the likelihood of their occurrence for the semiconductor devices are known. Their results for a representative memory requirement of 1K words × 16 bits under conditions of a single fault are shown in Table II-3. They observe that, for a one microsecond cycle time memory, there is no execution time penalty for chip-wide parity or Hamming code checkers because the overhead of 85 nanoseconds for each word can be overlapped. The variation in delay across the approaches appears to be sufficiently small that delay times should not be a major factor at this speed of memory operation (1 microsecond cycle time).

The results with respect to the Hamming and chip-wide parity code approaches are summarized in Table II-4. A comparison of the results shows that:

---

6 Jack, L.A., Kinney, L.L., Berg, R.O., "Comparison of Alternative Self-Checking Techniques in Semiconductor Memories", COMCON 77, pg. 170-173.

7. A totally self-checking circuit is a circuit that is self-testing for a normal input set, N, and a non-trivial fault set, F, and fault secure for N and a non-trivial fault set, $F_s$. A circuit is self-testing if, for every fault from a prescribed set, the circuit produces a non-code space output for at least one code input. A circuit is fault secure if, for every fault from a prescribed set, the circuit never produces an incorrect code space output for code space inputs.

TABLE II-3. MEMORY SELF-TEST METHODS COMPARISON CHART

| METHOD | MEMORY TYPE | MEMORY CONFIGURATION (WORDS x BITS) | IC COUNT |  |  |  | BOARD AREA (INCH$^2$) |  | ADDED MEMORY BITS | INCREASED MEMORY | EXECUTION OR DELAY TIME |
|---|---|---|---|---|---|---|---|---|---|---|---|
|  |  |  | MEMORY (INCL. EXTRA) | SELF-TEST | EXTRA MEMORY | TOTAL ADDED CIRCUITS | EXTRA MEMORY | SELF-TEST |  |  |  |
| 1) RAM No Self-Test | RAM | 256 x 4<br>1K x 1 | 17<br>16 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| 2) ROM No Self-Test | ROM | 1K x 4<br>512 x 8 | 4 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| 3) Single Bit Parity | RAM<br>ROM | 1K x 1<br>1K x 4 | 17<br>5 | 4<br>2 | 1<br>3 | 5<br>5 | 1<br>2.5 | 3<br>1.5 | 1K<br>1K | 6<br>6 | 120 ns<br>120 ns |
| 4) Byte-Wide Parity | RAM<br>ROM | 1K x 1<br>1K x 4 | 18<br>6 | 4<br>3 | 2<br>1 | 6<br>4 | 1.5<br>1 | 3<br>2.5 | 2K<br>2K | 13<br>13 | 120 ns<br>140 ns |
| 5) Chip-Wide Parity (4-bit) | RAM<br>ROM | 256 x 4<br>1K x 4 | 21<br>5 | 5<br>5 | 4<br>1 | 9<br>6 | 3<br>1 | 4<br>4 | 4K<br>4K | 25<br>25 | 85 ns<br>85 ns |
| 6) Chip-Wide Parity (8-bit) | ROM | 512 x 8 | 6 | 5 | 2 | 7 | 2.5 | 4 | 8K | 50 | 60 ns |
| 7) Memory Sum Check (Single Precision) | ROM | 1K x 4 | 7 | 0 | 3 | 3 | 2.5 | 0 | 1.4K | 9 | 7.14 ms |
| 8) Memory Sum Check (Double Precision) | ROM | 1K x 4 | 7 | 0 | 3 | 3 | 2.5 | 0 | 1.4K | 9 | 7.17 ms |
| 9) Sum Check Modified Double Precision Approach | ROM | 512 x 8 | 7 | 19 | 3 | 3 | 2.5 | 0 | 1.7K | 11 | 6.65 ms |
| 10) Hamming (21) Bit Double Error Detection | RAM<br>ROM | 1K x 1<br>1K x 4 | 21<br>6 | 11<br>6 | 5<br>2 | 16<br>8 | 4<br>1.5 | 8.5<br>4 | 5K<br>5K | 31<br>31 | 85 ns<br>85 ns |
| 11) Hamming (21) Bit Single Error Correction | RAM<br>ROM | 1K x 1<br>1K x 4 | 21<br>6 | 15<br>10 | 5<br>2 | 20<br>12 | 4<br>1.5 | 11.5<br>8 | 5K<br>5K | 31<br>31 | 100 ns<br>100 ns |
| 12) Hamming (22) Bit Triple Error Detection | RAM<br>ROM | 1K x 1<br>1K x 4 | 22<br>7 | 14<br>8 | 6<br>3 | 20<br>11 | 4.5<br>2.5 | 10.5<br>6 | 6K<br>6K | 38<br>38 | 70 ns<br>70 ns |
| 13) Hamming (22) Bit Double Error Detection, Single Error Correction | RAM<br>ROM | 1K x 1<br>1K x 4 | 22<br>7 | 18<br>12 | 6<br>3 | 24<br>15 | 4.5<br>2.5 | 13.5<br>9.5 | 6K<br>6K | 38<br>38 | 120 ns<br>120 ns |

TABLE II-4. SUMMARY IMPLEMENTATION TRADE-OFF COMPARISON

| SELF-[illegible] | [illegible] TYPE | [illegible] | [illegible] MEMORY | [illegible] | [illegible] | [illegible] CASE COVERAGE | |
|---|---|---|---|---|---|---|---|
| | | | | | | [illegible] | [illegible]CED ADDRESS |
| Chip-Wide Parity | RAM<br>ROM | 4 | 40%<br>30% | 25%<br>25% | 0<br>0 | 100%<br>100% | 100%<br>100% |
| Hamming-Like Code<br>5 Bit | RAM<br>ROM | 15<br>- | 60%<br>60 | 31%<br>31% | 0<br>0 | 100%<br>100% | 100%<br>100% |

1) Both chip-wide and Hamming-like codes provide extremely high coverage for massive on-chip failure modes.

2) Both Hamming-like codes and chip-wide parity achieve 100% error detection for either single device data or address faults for chips no wider than 4 bits.

3) Hamming-like codes give better detection for multiple device failures than simple parity schemes at an increase in hardware.

4) Chip-wide parity requires fewer parity bits, less detection circuitry, and has less interconnect complexity.

5) Neither approach permits errored data or instructions to be passed to the CPU assuming single chip failure.

Of the above approaches, only Hamming-like codes have the potential for error correcting capability. Techniques for designing error-free decoding, coupled with error correcting memory, have been described by Carter, et al[8]. Single error correcting double error detecting (SEC/DED) Hamming-like codes have been used to protect memories where it is assumed that single failures affect one bit of the word and two failures affect two bits of the retrieved word. Early versions of this approach used "self-testable" SEC/DED decoders and encoders or translators for converting from bus parity code to memory coding and vice versa. Here self-testing is understood to mean circuits that test the proper functioning of every component during normal operation. The decoding circuitry is dynamically tested while it performs its function of correcting erroneous data without mistaking these errors for errors caused by circuit faults and vice versa. The results of applying these techniques to memories of 32, 64, and 128 bits are shown in Table II-5 from Carter, et al. The actual data bits are listed in the column labeled k and the redundant or check bits required are shown under the r column (n is the sum of K + r). The next column to the right, labeled "Conventional SEC/DED to Byte Parity Circuits", lists the circuits needed to translate from the SEC/DED memory code to the bus parity code using conventional design techniques. The next column to the right, labeled "Self-Checking SEC/DED to Byte Parity Circuits", gives the figures for the self-testing version (including translation). As seen in the last column, only a small increase was required to achieve self-checking and this difference decreases with word length. Hence, translators can be a source of large implementation cost and either should be avoided if possible or should be minimized by choosing compatible codes to the extent possible.

8 Carter, W.C., Jessep, D.C., and Wadia, A., "Error-Free Decoding for Failure Tolerant Memories, FTC 71.

TABLE II-5. COMPARISON OF CONVENTIONAL AND SELF-CHECKING MEMORY IMPLEMENTATIONS

| n | k | r | CONVENTIONAL SEC/DED TO BYTE PARITY CIRCUITS | SELF-CHECKING SEC/DED TO BYTE PARITY CIRCUITS | % INCREASE |
|---|---|---|---|---|---|
| 32 | 26 | 6 | 1050 | 1130 | 7.4 |
| 64 | 57 | 7 | 2300 | 2380 | 3.5 |
| 128 | 120 | 8 | 4950 | 5000 | 1.1 |

n = Total Number of Bits in Message Being Protected

k = Total Number of Information Bits in Message Being Protected

r = Total Number of "Check" Redundancy Bits In Message Being Protected

n = k+r

This work has led to the development of techniques for designing self-checking circuits not only for memories but for all computer functions, such as the arithmetic and logic unit and control. In the case of memories, Carter and McCarthy[9] have extended these principles and have designed a fault-tolerant memory based on a modified Hamming-like SEC/DED code. The logic of this memory is designed to be fault secure, self-testing, and is claimed to exhibit good cost/performance. The testing procedures are designed to detect faults and prevent error accumulation. In the recovery process, single error correction can be validated and most double errors caused by two faults corrected.

Husband and Szygenda[10] have provided a detailed synthesis and analysis of a cost effective, ultrareliable, high speed, semiconductor memory system. A 16K word by 64 bit memory system with a 250 nanosecond cycle time was designed that corrected over 99% of all single faults. The approach is based on a single error Hamming code, and support electronics that are designed to cause all faults outside the memory proper to produce no more than one bit error in any one memory word. The error-correcting circuits make up less than 2% of the total circuitry and the increase in circuitry over the simplex system is less than 20%. Cycle time is not increased unless a fault occurs. The implementation results are tabulated in Table II-6 taken from Husband and Szygenda's paper.

9 Carter, W.C., McCarthy, C.E., "Implementation of Experimental Fault Tolerant Memory, IEEE Trans. on Computer, C-25, No. 6, June 1976.

10 Husband, E. W., Szygenda, S.A., "Synthesis and Analysis of a Cost Effective, Ultrareliable, High Speed, Semiconductor Memory System, IEEE TC on Reliability, Vol. R-25, No. 3, August 1976.

TABLE II-6

COMPARISON OF 16K BY 64 BITS

SIMPLEX AND FAULT TOLERANT MEMORY SYSTEM

| | SIMPLEX SYSTEM | FAULT TOLERANT SYSTEM | INCREASE IN DEVICES | PERCENTAGE OF INCREASE |
|---|---|---|---|---|
| Memory | 4096 | 4608 | 512 | 12.5 |
| Address | 48 | 128 | 80 | 167 |
| Write Enable | 6 | 16 | 10 | 167 |
| Chip Select | 10 | 31 | 21 | 210 |
| Data In | 8 | 9 | 1 | 12.5 |
| Data Out | 40 | 45 | 5 | 12.5 |
| Error Correction | | 96 | 96 | |
| Total | 4208 | 4933 | 725 | 17.2 |

To summarize, the particular memory protection approach is a function of the memory cycle time, the word length, the protection used in the rest of the system, and the size (capacity) of the memory. Assuming a system compatible memory cycle time and a memory device selection based, primarily, on minimum cost the recommended protection approach as a function of memory size is given in Table II-7.

TABLE II-7

RECOMMENDED MEMORY PROTECTION TECHNIQUE

| SMALL | MEDIUM | LARGE |
|---|---|---|
| 1) Replication | 1) SEC DED Hamming-like codes with self-checking encoders decoders using single bit (word) memory device | 1) SEC DED Hamming-like codes with self-checking encoders, decoders using single bit (word) memory device |
| 2) Triplication with voters - TMR | 2) Duplication and compare with checking by chip-wide parity using b-bit (word) memory device | |

## 5. Control

Until recently, replication was the only known method of control unit error detection[11]. However, the introduction of microprogrammed techniques to control unit design has eased the problem since the complexity of the unit is reduced. This has led to the study of low-cost techniques for the detection of control unit errors. Toy, et al[12] designed a self-checking microprogrammed control unit based on a combination of parity checking, bit compare and interleaving. It has been shown that this design can be made totally self-checking with respect to single faults. Diaz[13] showed that totally self-checking concepts could be applied to synchronous sequential machines (Moore type) in addition to the combinatorial circuits considered earlier assuming that the clock line is fault-free. Later, Ozguner[14] developed approaches for designing totally self-checking asynchronous sequential machines. Ho[15] describes the design of totally self-checking computers including the microprogrammed control unit. This machine is designed to halt upon the detection of a fault and can be instrumented to provide fault location information to within a few gate levels. Ashjaee and Reddy[16] describe totally self-checking checkers for separable codes and point out that, for certain designs and separable codes, the corresponding checkers cannot be realized. For Type-I checkers (see Reddy) of totally self-checking systems, they define sufficient conditions on separable codes that insure that the checker can be realized.

Unfortunately, most of the work is based on a single stuck-at-one or stuck-at-zero-error model for the checker. Consequently, it is not compatible with an LSI implementation of the checker. However, a lower level of integration implementation, such as MSI or SSI, would probably meet the single error model. Such an approach would complement many bit-slice microprocessor control units since they are usually implemented with relatively low-level integration devices combined with high-speed, highly integrated memory devices, e.g., read-only memory (ROM) and programmable read-only memory (PROM).

---

11 Eckert, J.P., Weiner, J.R., Welsh, H.F., Mitchell, H.F., "The UNIVAC System", AIEE-IRE Conf. 6-16, 1951.

12 Toy, W.N., "Modular LSI Control Logic Design With Error Detection", IEEE TC-20(2), 1971, pg. 161-162.

13 Diaz, M., "Design of Totally Self-Checking and Fail Safe Sequential Machines", Proc. Fourth Annual International Symposium on Fault Tolerant Computing, June 1974, pg. 3-19 - 3-24.

14 Ozguner, F., "Design of Totally Self-Checking Asynchronous Sequential Machines", Coordinated Science Laboratory Report R-679, Univ. of Illinois, May 1975.

15 Ho, D.S., "The Design of Totally Self-Checking Systems", PhD Thesis, Univ. of Illinois, 1976.

16 Reddy, J. M., Ashjaee, M.J., "On Totally Self-Checking Checkers for Separable Codes", IEEE TC, Vol. C-26, No. 8, August 1977.

Perhaps more importantly, none of the coding approaches investigated resulted in low-cost implementations. When all the factors were considered, the implementation costs were in excess of duplication and introduce additional delays in the control loop which, time-wise, were already the limiting path.

For these reasons, replication is recommended for the bit-slice microprocessor control unit. For systems that require essentially uninterrupted processing, a form of triplication is recommended. Assuming that the LRU could be as large as the simplex control unit, augmented TMR would provide sufficient error location resolution, so that resource costs would approximate that of triplication plus the voters. Speed would be reduce only slightly due to the added delays of the voters.

Bit-slice based microprocessor systems that can tolerate "short" interruptions have the option of considering either replication or some form of coding. However, even here, replication is favored because real-time error masking of error correction codes is not needed and replication provides at least one identical copy of the unfailed structure after the occurrence of the error.

For monolithic microprocessors, the control unit is considered as a part of the CPU and the recommendations made in the processor section apply. Some form of TMR is recommended.

## 6. Buses

Bus protection depends on the buses and interface failure modes. Military standards, such as MIL STD 1553A, and the environment strongly affect the bus protection approaches that can be considered for a particular application. The source and sink of the information transmitted over the bus also strongly affect the protection approach because if the code employed at either end of the bus is different from that used by the bus, a code translator may be required. If all three employ different coding schemes, two translateors are required. Depending on the code pair, the translator may be quite complex and itself require protection. It, therefore, behooves the system designer to utilize as few different codes throughout the system as possible. Where different codes are required they should be selected to minimize the translator requirements, and vice versa if two-way communication is to be maintained across the bus.

For the applications considered here, where the systems are small and the number of devices is of the order of 100 or less, the processor buses should be short and the interface requirements should dominate. Thus, the problem can be viewed as just an extension of the design of protected logic. For monolithic microprocessors, which are most likely to employ some version of TMR, the processor internal buses are protected by TMR when the functional units, i.e., processor, memory, are viewed as one logic unit; the buses are indistinguishable from other signal paths.

For bit-slice microprocessors, the buses between the processor, memory, and control can be viewed as indistinguishable for many applications. Thus, the same considerations drive the bus protection problem as the functional units and the same techniques can be employed. Since the control and processor units will quite likely be implemented using some form of replication, the nature of the problem will be one of transitioning between various forms of duplication and triplication. The memory to the control and processor bus problem is different for a number of reasons. First, as just indicated, the processor and control will, more than likely, be protected by some form of replication, while the memory will, most likely, be protected by some conventional code, like Hamming. Second, the number of buses can vary between one and three, depending on whether the communication is split between an input or output bus and or whether the bus is segregated by function between address and data. Lastly, the treatment of the control signals in the receiving units can vary between checking whether the signal has been correctly transmitted to whether the signal is properly received and returning a signal to the control for reconfiguration, indicating the results of the control transmission -- error non error.

7. Clock

The reliable generation and distribution of timing signals is needed to insure proper operation of synchronous sequential circuits in self-diagnosing processors. Errors on these timing or clock lines may be due to interconnection failures or malfunctions in the source of the signal. For errors produced by malfunctions in the generation mechanisms, errors can be classified as either catastrophic, which is equivalent to a stuck-at fault on the line, or variational as in a frequency change. Further classification has been established based on the circuit used to monitor the line. It is:

1) Discrete, charge discharge circuits

2) Retriggerable monostable multivibrator circuits

3) Digital counter circuits

4) Integrator circuits

None of the schemes can check the input for all possible errors without resorting to duplication and each of the circuits is susceptible to undetectable internal faults. Usas[17] describes a self-checking periodic signal checker as shown in Figure II-1. It uses the same hardware as a duplication scheme using a retriggerable monostable circuit, but it is self-testing and the duplicated design is not. Additionally, this checker also detects duty cycle errors. Since M1 and M2 are arranged to run 180$^{0}$ out-of-phase, the circuit detects unidirectional errors and the two monostables can be realized in a single integrated circuit package without concern for failures affecting the common power and ground distribution to the individual circuits. It is recommended, for effective error detection, that the checker be wired to a memory element or clocked module following the last fanout branch. This permits the detection of faults on any of the fanout points in the wiring of the clock line.

17 Usas, A.M., "The Detection of Errors in Periodic Signals", Technical Note #45, Stanford University, April 1974.

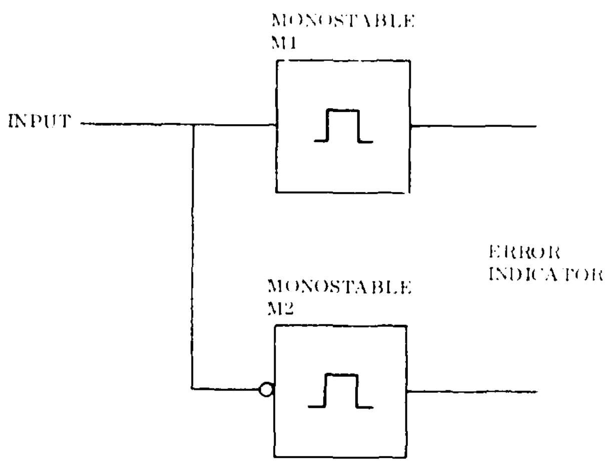


Figure II-1. Totally Self-Checking Periodic Signal Checker

The error indicator is designed to complement the checker and provide visual output for both fixed and momentary errors, as shown in Figure II-2. It is not self-testing but is fault secure with respect to all faults affecting only a single flip-flop.

Other approaches utilize an array of identical oscillator modules to produce a number of phase-locked clock signals. A technique commonly used is majority vote among 2f + 1 redundant signals, which produces a valid output if the redundant inputs are suitably synchronized. Daly[18] shows that the simple majority function is insufficient and that "gliches" or sliver pulses can result because the output depends on the failed elements during part of the clock period. He shows that by incorporating hysteresis as in-line

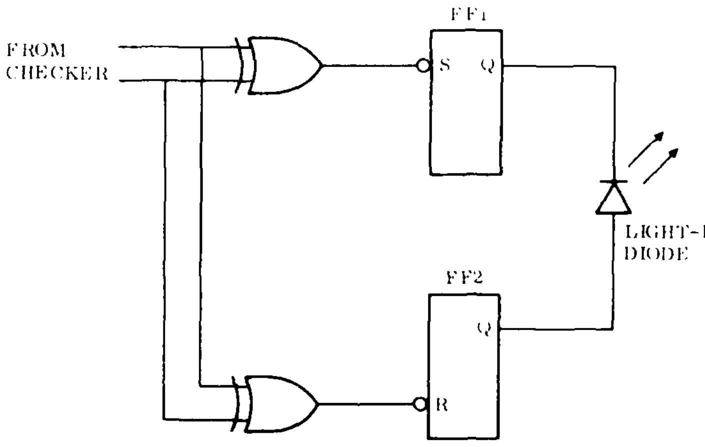


Figure II-2. Fail Safe Error Indicator

18 Daly, W.M., "A Fault Tolerant Digital Clocking System", FTC 73, pg. 17-22.

receivers, the difficulty can be eliminated. Figure II-3 and II-4 show one of four identical elements of a fault-tolerant clock system module that will tolerate the failure of any one element using conventional TTL integrated circuits. The circuitry of Figure II-3 is designed to receive and vote on four clock outputs, A, B, C, and D. It generates the majority functions and produces pulse outputs corresponding to the falling and leading edges of the majority function. Z10 is a monostable multivibrator that determines the clock frequency. Z11 is a one-shot that estaolishes a lower limit on switching time.

A clock receiver (hysteresis voter) consists of the logic of Figure II-3 plus a flip-flop that is set by DRSTA and reset by DSETA. The flip-flop output is the desired synchronized clock signal and drives the user circuitry.

Another intrinsic clock approach employs oscillator standby redundancy and majority voting with hysteresis. Each oscillator contains switching circuitry to select one of the three oscillators to develop the redundant clock signals as shown in Figure II-5. Operation of the circuit is such that if oscillator A driving the majority gates fails, the detector causes its latch to set, causing the switchover logic to select the next available oscillator. If oscillator B's latch is not set, it provides the clock signal for the system. If, however, B's latch is also set, then oscillator C is selected by the crossover switch to drive the majority gates. By providing external control of the latches, switchover can be commanded by the using system for reasons such as excessive frequency drift and testing.

The reliability model of the system is shown in Figure II-6. It can be seen that, rather than the conventional redundant voted approach (Figure II-6a) where two out of three oscillators are required for success, only one out of three is required (Figure II-6b). Although the added circuits in the oscillator chain slightly increase the failure rate of the A and B oscillator channels, the net effect is increased reliability because of the increased tolerance; only one oscillator is necessary for successful operation.

For the self-diagnosing applications, a combination of the standby redundant oscillators and the fault-tolerant receiver, similar to that described (Figure II-4 plus a flip-flop) is recommended for synchronization. An error indicator circuit should be added for failed synchronizer error location indication.

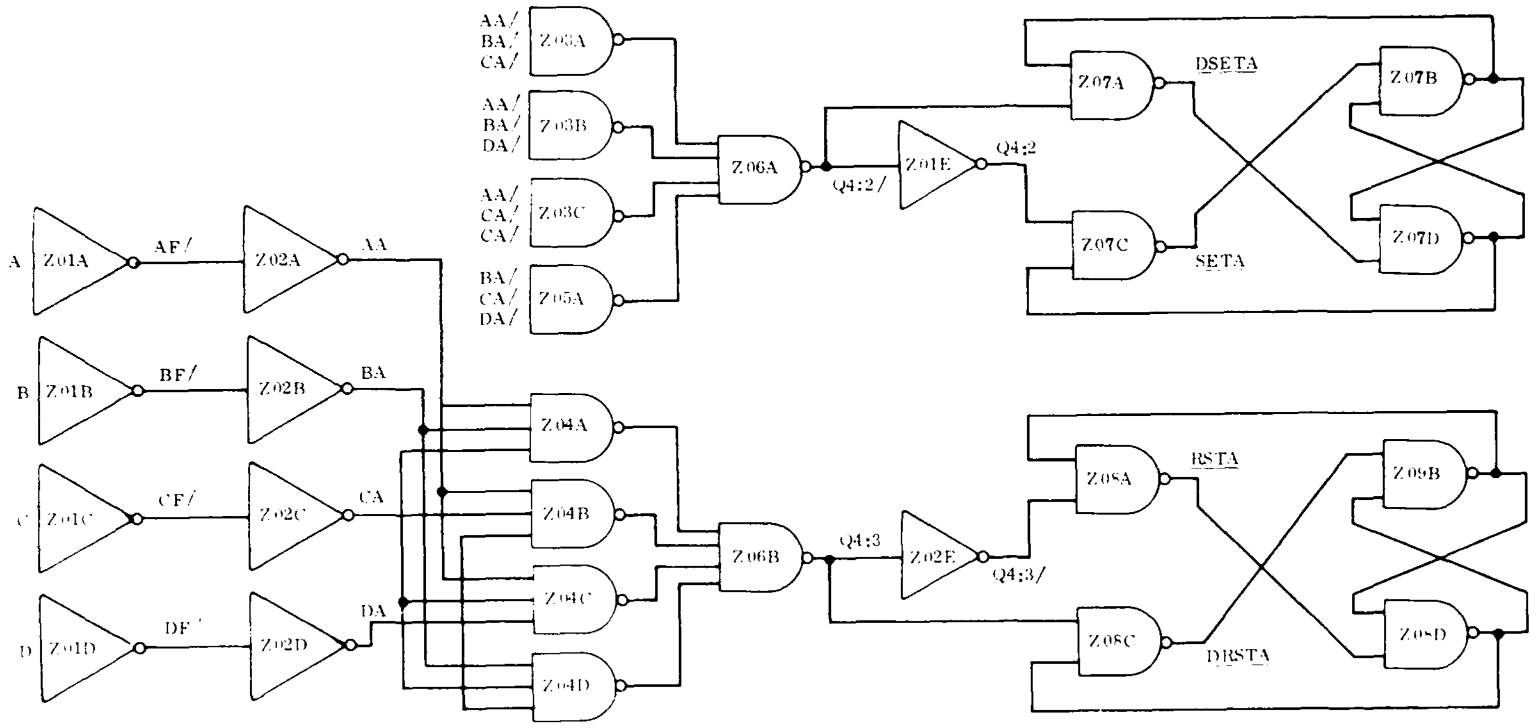


Figure II-3. Clock Receiver (Hysteresis Voter)

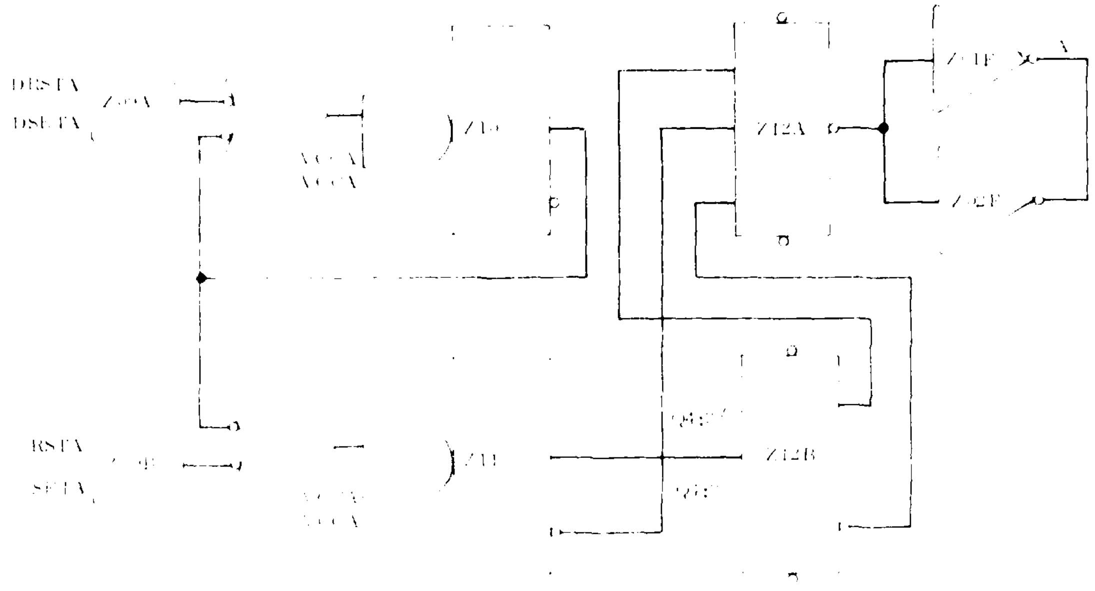


Figure II-4. Output Circuit of a Clock Element

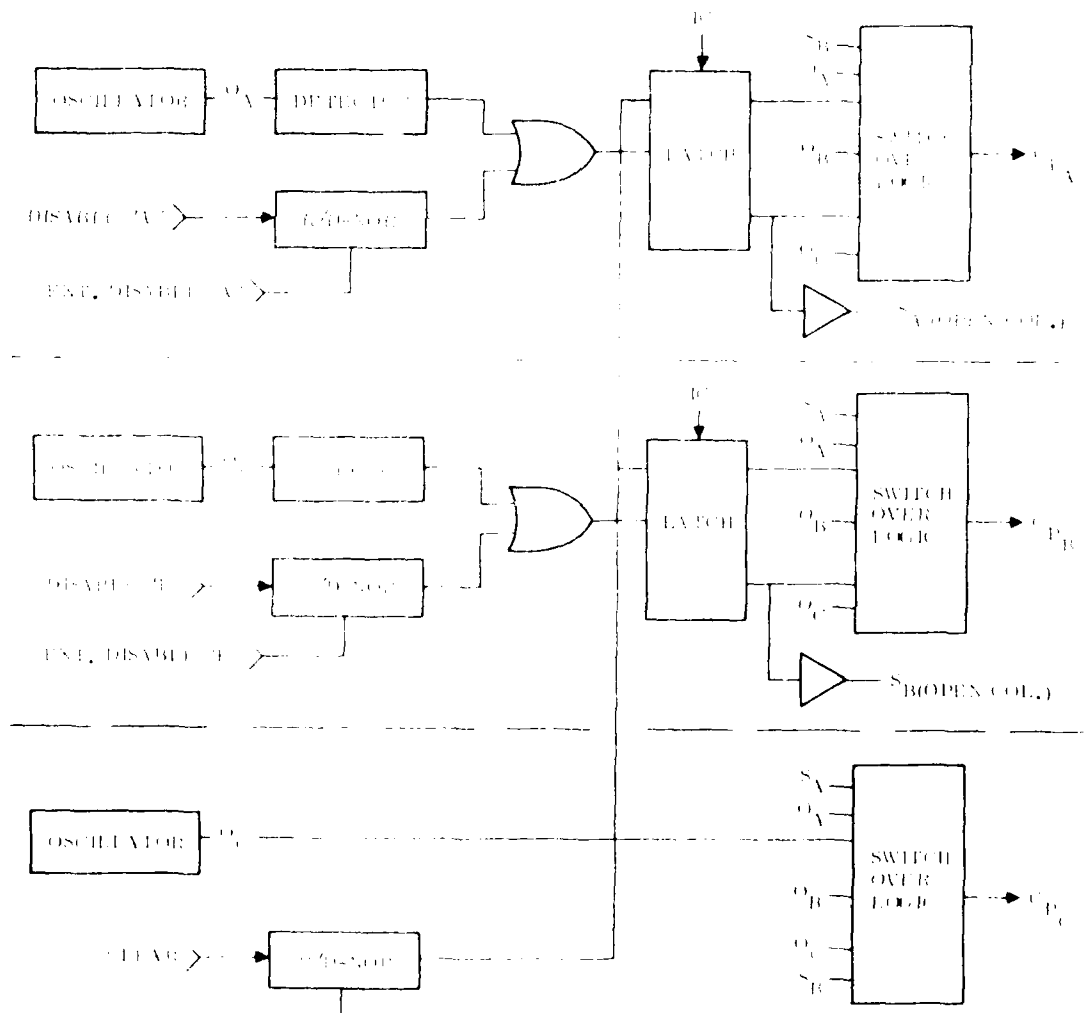
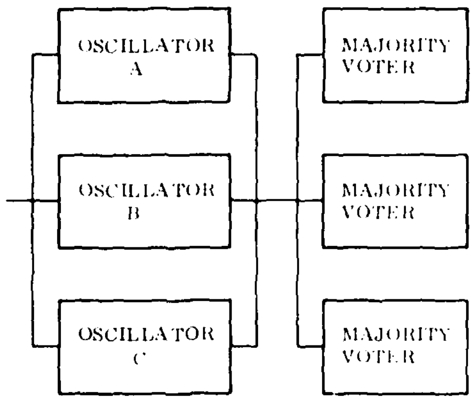
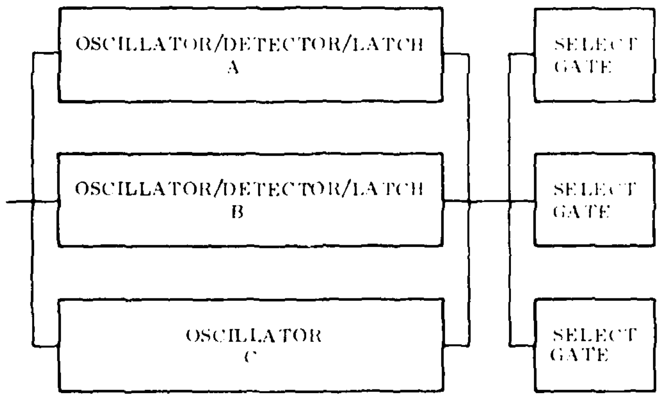


Figure II-5. Standby Redundant Oscillator



a. PRESENT MODEL



b. PROPOSED MODEL

Figure II-6. Oscillator Reliability Model Block Diagram

Distribution List

AFAL-TR-78-183

Self-Diagnosing Design Techniques

3 AFEWC/ESRI (Mandatory)
San Antonio TX 78243

1 AFSC/IN (Mandatory)
Andrews AFB DC 20334

10 AFWAL/AADE
(Capt J. B. Rawlings)
WPAFB OH 45433

1 AFWAL/POTC
(Charles E. Ryan, Jr.)
WPAFB OH 45433

[illegible] AFWAL/TST (STINFO)
WPAFB OH 45433

1 AFWL/NSCA
(Major McClellan)
Kirtland AFB NM 87117

1 AUL/LSE (Mandatory)
Maxwell AFB AL 36112

[illegible] DTIC (Mandatory)
Cameron Station
Alexandria VA 22314

5 General Electric Corporation
Electronics Laboratory [illegible]
Electronics Park
Syracuse NY 13221

1 SRI International
ATTN: Tom Hodges, Bldg 320
333 Ravenswood
Menlo Park CA 94025

1 USAF/SAMID (Mandatory)
Washington DC 20330

ATE
LMED
8